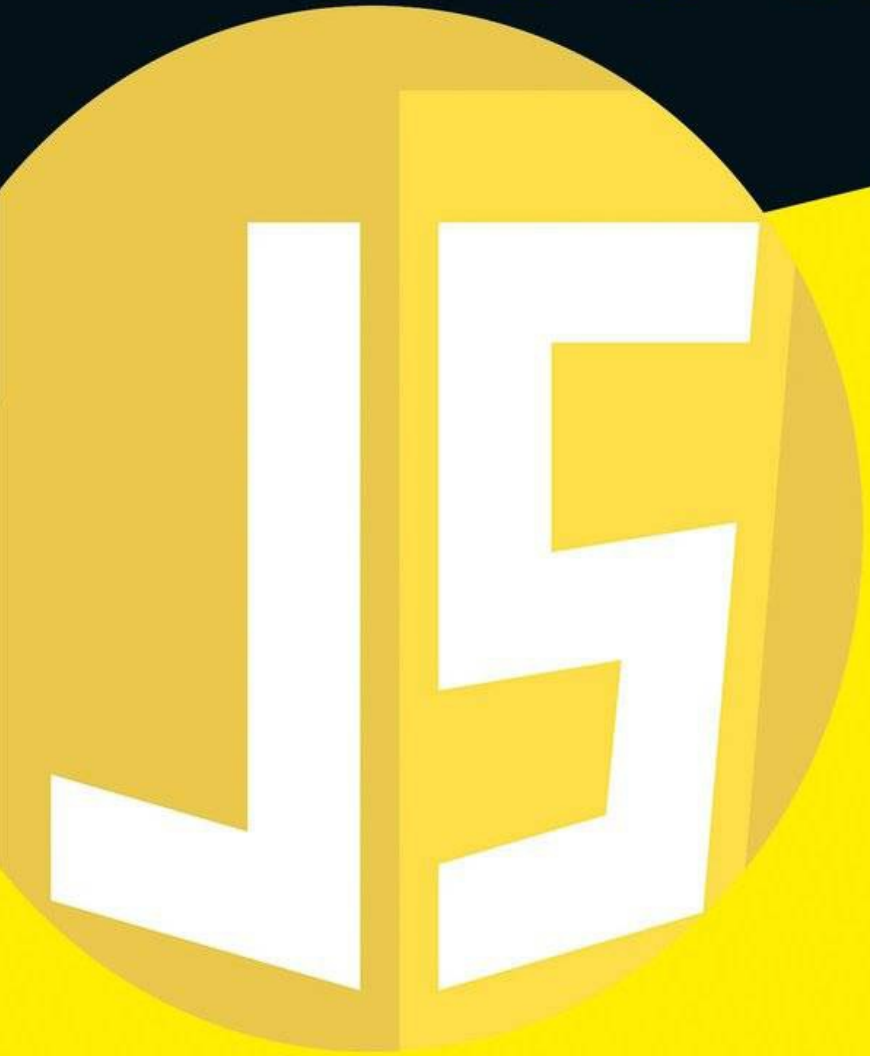


Avec les Nuls, tout devient facile !

2e édition

JavaScript pour les nuls



- Travailler avec les variables, les tableaux, les opérateurs et les instructions
- Manipuler des documents avec le DOM
- Formulaires HTML et CSS
- Ajax et JSON
- HTML5 et ses API
- jQuery

Chris Minnick
Eva Holland



JavaScript

pour les nuls

2e édition

Chris Minnick et Eva Holland

FIRST Interactive

JavaScript pour les Nuls, 2e édition

Titre de l'édition originale : *Coding with JavaScript for Dummies*
Copyright © 2015 Wiley Publishing, Inc.

Pour les Nuls est une marque déposée de Wiley Publishing, Inc.
For Dummies est une marque déposée de Wiley Publishing, Inc.

Édition française publiée an accord avec Wiley Publishing, Inc.
© Éditions First, un département d'Édi8
12, avenue d'Italie
75013 Paris – France
Tél. : 01 44 16 09 00
Fax : 01 44 16 09 01
Courriel : firstinfo@editionsfirst.fr
Site Internet : www.pourlesnuls.fr

ISBN : 978-2-412-02888-9
ISBN numérique : 9782412032916
Dépôt légal : août 2017

Mise en page : Pierre Brandeis

Tous droits réservés. Toute reproduction, même partielle, du contenu, de la couverture ou des icônes, par quelque procédé que ce soit (électronique, photocopie, bande magnétique ou autre) est interdite sans autorisation par écrit d'Édi8.

Cette œuvre est protégée par le droit d'auteur et strictement réservée à l'usage privé du client. Toute reproduction ou diffusion au profit de tiers, à titre gratuit ou onéreux, de tout ou partie de cette œuvre est strictement interdite et constitue une contrefaçon prévue par les articles L 335-2 et suivants du Code de la propriété intellectuelle. L'éditeur se réserve le droit de poursuivre toute atteinte à ses droits de propriété intellectuelle devant les juridictions civiles ou pénales.

Ce livre numérique a été converti initialement au format EPUB par Isako www.isako.com à partir de l'édition papier du même ouvrage.

Introduction

JavaScript, c'est chaud ! Celui qui a débuté comme un langage « rapide et facile à apprendre et à utiliser » créé pour l'un des premiers navigateurs Web est devenu le langage de programmation le plus populaire au monde. Et la demande de programmeurs JavaScript continue à croître en permanence.

Ce livre est votre point d'entrée pour apprendre, comprendre et devenir efficace avec les concepts clés de JavaScript. Que votre but soit de vous (re)convertir professionnellement dans la programmation en JavaScript, ou plus simplement de rendre votre propre site Web plus interactif, vous pouvez être assuré que le contenu et les techniques présentés dans ce livre sont en phase avec les standards et les meilleurs pratiques JavaScript les plus récents.

Couplé avec de nombreux exercices, à effectuer vous-même ou à télécharger en ligne, chaque chapitre contient des exemples de code réel que vous pouvez essayer et tester chez vous dans votre navigateur Web préféré (plusieurs, c'est encore mieux).

Pour devenir un champion de tennis, il n'y a pas de mystère : il faut s'entraîner, s'entraîner et encore s'entraîner. C'est exactement pareil pour la programmation. Pour devenir un champion de JavaScript, il faut coder, coder et encore coder !

À propos de ce livre

Ce livre est un guide amical et abordable pour commencer à écrire du code JavaScript. Comme langage de programmation, JavaScript est plutôt facile à appréhender. Il est tout à fait possible de commencer très vite à l'utiliser. Du fait qu'il est si accessible, de nombreuses personnes ayant débuté comme auteurs de pages Web se sont retrouvées dans la position de devenir responsables de l'écriture, de la modification et de la maintenance de code JavaScript. Si c'est ce que vous recherchez, ce livre devrait vous permettre de vous mettre rapidement au travail.

Que vous ayez déjà ou non une petite expérience de JavaScript, ce livre vous montrera comment écrire du « bon » code JavaScript.

Les sujets abordés dans le livre couvrent de multiples sujets, notamment :

- Comprendre les structures de base des programmes JavaScript
- Intégrer JavaScript avec HTML5 et CSS3
- Structurer vos programmes à l'aide de fonctions
- Travailler avec les objets JavaScript
- Utiliser des techniques JavaScript avancées, comme les objets, Ajax, les rappels et les fermetures
- Aborder jQuery

Apprendre JavaScript, ce n'est pas seulement apprendre la syntaxe d'un langage. C'est aussi accéder à une horde d'outils, et à la communauté qui s'est construite autour de ce langage. Les programmeurs professionnels ont largement fait évoluer les outils et les techniques utilisés en JavaScript tout au long d'une histoire déjà longue et riche. Tout au long de ce livre, nous mettrons en exergue les bonnes pratiques, ainsi que les outils servant à tester, documenter et écrire plus rapidement du meilleur code !

Pour rendre le livre plus facile à lire, nous utiliserons quelques conventions :

- Tout le code JavaScript, ainsi que les balises HTML et CSS, sera mis en évidence ainsi :

```
document.write("Hello !");
```

- Les marges d'une page de livre n'offrent pas le même espace que la surface de votre écran. C'est pourquoi de longues lignes de code HTML, CSS et JavaScript peuvent se trouver réparties sur plusieurs lignes de texte. Mais n'oubliez pas que, pour votre ordinateur, il s'agit dans ce cas d'une seule et même ligne. Pour éviter la confusion, nous ajouterons dans ce cas une indentation au début de la ou des lignes suivantes, comme ici :

```
document.getElementById("unElementDansLeDocument").
        addEventListener("click",faireQQChose,false);
```

- HTML et CSS ne sont pas réellement soucieux de savoir si vous utilisez des majuscules, des minuscules ou une combinaison des deux. JavaScript, par contre, est très pointilleux à ce sujet. Pour obtenir les résultats voulus avec les exemples de code de ce livre, veillez à respecter scrupuleusement l'utilisation des majuscules et des minuscules dans tout ce code.

» De même, les noms des fonctions, variables et autres, ne sont pas traduits afin d'éviter aussi bien les risques de confusion que d'erreur. Cela ne devrait pas être bien gênant, sachant que, de toute manière, tous les langages de programmation, et donc JavaScript, ont pour langue maternelle l'anglais (ou plutôt l'anglais américain).

Quelques suppositions stupides

Vous n'avez absolument pas besoin d'être un petit génie de l'informatique, ou un hacker de haut vol, pour comprendre la programmation. Vous n'avez pas non plus besoin de comprendre comment fonctionne la tuyauterie de votre ordinateur. Et il n'est même pas nécessaire de savoir comment compter en binaire.

Pour autant, il faut bien que nous puissions faire quelques suppositions sur vous. Nous partirons donc de l'idée que vous savez allumer un ordinateur, utiliser un clavier et une souris, que vous avez une connexion Internet, et que vous savez ouvrir un navigateur Web. Et si vous avez déjà quelques connaissances sur la manière de créer des pages Web (dont notamment des rudiments de HTML), ce sera parfait pour débuter.

Les autres choses que vous devez savoir pour écrire et exécuter du code JavaScript sont explicitées dans le livre. Et vous comprendrez très vite que l'une des clés en programmation, c'est l'attention portée aux détails.

Icônes utilisées dans ce livre

Voici une liste des icônes dont nous nous servons dans ce livre pour mettre en évidence certains points particuliers :



Cette icône signale des trucs et astuces, ou encore des raccourcis qui vous permettront d'épargner temps et efforts.



C'est quelque chose auquel vous devez prêter une attention particulière et que vous ne devez pas oublier.



Faites attention, très attention. Cette icône pointe le doigt sur des problèmes ou des pièges à éviter.



Il s'agit de détails techniques qui peuvent ou non vous intéresser. Si vous avez des tendances *geek*, n'hésitez pas à lire ces compléments. Sinon, vous avez parfaitement le droit de passer votre chemin.

Où trouver les exemples du livre ?

Vous pourriez bien entendu retaper de A à Z tous les exemples de code de ce livre. Mais pourquoi se fatiguer inutilement, et de plus devoir corriger de multiples erreurs de frappe, alors qu'il est très facile de télécharger tout le matériau dont vous avez besoin.

Vos points d'entrée sont les suivants :

Dans votre navigateur Web, rendez-vous à l'adresse suivante :

http://www.pourlesnuls.fr/

Cliquez sur l'icône de loupe et commencez à saisir *JavaScript*, puis cliquez sur l'icône de couverture du livre dans la liste qui s'affiche. Dans la page du livre, cliquez sur l'icône Télécharger. Le fichier à télécharger est petit. Tout devrait donc aller très vite.

Un site compagnon est disponible à l'adresse suivante :

http://www.dummies.com/go/codingwithjavascript

Vous y trouverez une description du livre, ainsi que de nombreux liens vers le site qui vous permet de tester les exemples en « live ».

Un site plus spécifique est dédié à ce livre, à l'adresse :

http://www.codingjsfordummies.com/

Vous pourrez non seulement y télécharger le code original (non traduit) des exemples du livre, mais aussi y trouver des compléments, voire des correctifs.

Enfin, pour accéder à des articles en ligne (et en anglais, bien sûr) qui devraient vous aider à parfaire vos connaissances, ou encore vous proposer des mises à jour, vous pouvez visiter la page :

http://www.dummies.com/extras/codingwithjavascript

Et maintenant ?

Coder avec JavaScript est en fait un vrai plaisir. Une fois que vous aurez chargé un peu de connaissances dans votre besace, le monde des applications Web interactives vous appartiendra !

Nous espérons que vous apprécierez ce livre et que vous aurez envie d'aller encore beaucoup plus loin une fois que vous l'aurez refermé.

Bonne programmation en JavaScript !

Débuter avec JavaScript

DANS CETTE PARTIE…

Découvrir comment écrire votre premier programme JavaScript

Découvrir comment travailler avec des variables et des tableaux

Découvrir comment travailler avec des opérateurs, des expressions et des instructions

Utiliser des boucles et des branchements dans votre code JavaScript

Trouver les exemples du livre et d'autres ressources sur le Web

Chapitre 1

JavaScript, le langage le plus incompris du monde

DANS CE CHAPITRE :

- **Qu'est-ce qu'Arduino ?**
- **Apprendre à connaître JavaScript**
- **Comprendre ce que fait JavaScript**
- **Comprendre pourquoi vous avez besoin de JavaScript**

« Les gens me comprennent si mal qu'ils ne comprennent même pas pourquoi je me plains de ce qu'ils ne me comprennent pas. »

Søren Kierkegaard

JavaScript n'a pas toujours été considéré comme il l'est aujourd'hui. Il y a des gens qui l'ont appelé le meilleur et le pire langage de programmation du monde. Mais, notamment ces dernières années, de nombreuses améliorations ont été apportées aussi bien à la manière dont les développeurs écrivent du code JavaScript qu'aux interpréteurs JavaScript eux-mêmes. Ces améliorations font que JavaScript est aujourd'hui un bien meilleur langage qu'il ne l'était dans sa jeunesse.

Dans ce chapitre, vous allez découvrir ce qu'est JavaScript, ainsi que quelques notions sur l'histoire de ce langage. Vous y verrez également ce que fait JavaScript et pourquoi vous avez besoin de le connaître.



N'oubliez pas de visiter le site Web compagnon de ce livre pour y télécharger les exemples développés ici !

JavaScript, c'est quoi ?

Au tout début du Web, les navigateurs n'étaient que de simples lecteurs, comme l'illustre la Figure 1.1. Ils ne possédaient aucune capacité particulière, si ce n'est celle d'afficher du texte dans différentes tailles. Du jour où Microsoft publia Internet Explorer, la guerre des navigateurs commença à faire rage, et leurs fonctionnalités se développèrent très vite. L'un offrit la possibilité d'afficher des images, puis un autre d'utiliser des polices de caractères différentes, puis de faire clignoter le texte, de le déplacer, et toutes sortes d'autres possibilités nouvelles à l'époque.

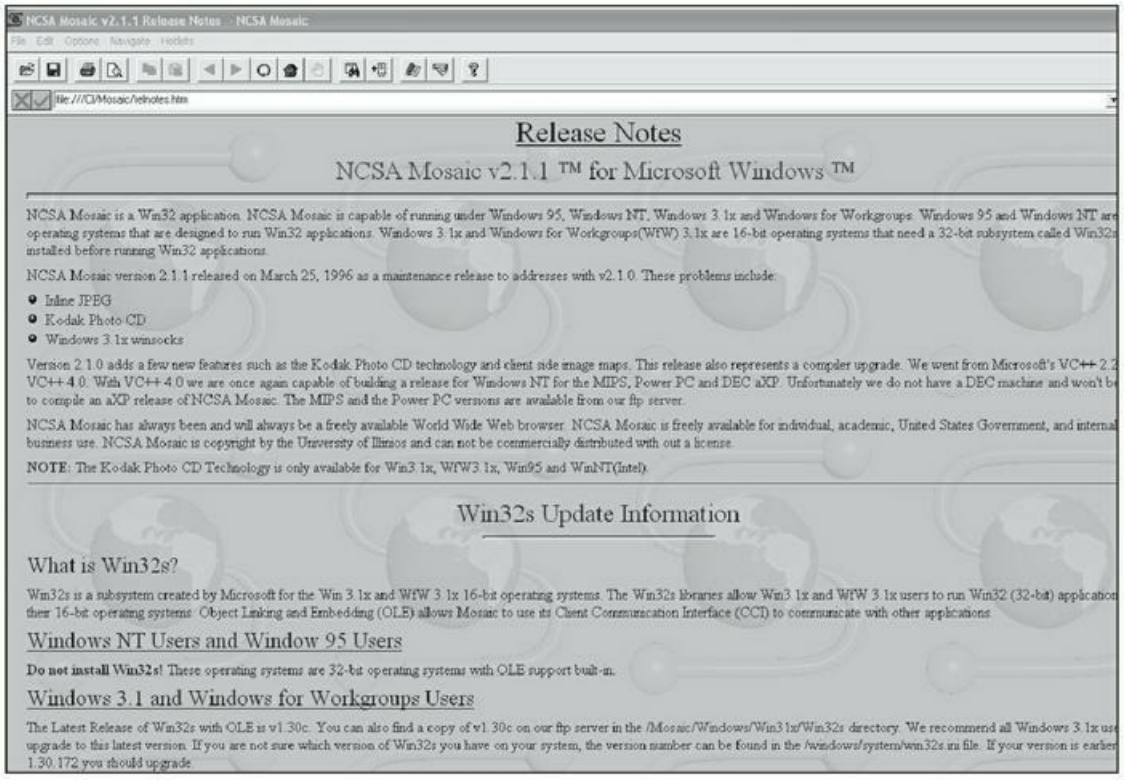

NCSA Mosaic v2.1.1 Release Notes - NCSA Mosaic
Release Notes
NCSA Mosaic v2.1.1 ™ for Microsoft Windows ™
Win32s Update Information
What is Win32s?
Windows NT Users and Window 95 Users
Windows 3.1 and Windows for Workgroups Users

FIGURE 1.1 : Les premiers navigateurs Web n'étaient pas jolis à regarder.

Très vite, l'idée que les navigateurs devraient être capables de gérer les choses eux-mêmes, plutôt que de se comporter comme des programmes capables d'afficher des documents d'une manière plus ou moins décorative, s'imposa à tout le monde.

La comète Eich

La naissance de JavaScript se situe en 1995 dans les bureaux de Netscape. Le créateur de JavaScript, Brandon Eich, écrivit celui-ci en un temps record (la légende dit que cela lui prit moins de dix jours !). Pour y arriver, il emprunta les meilleures fonctionnalités de divers autres langages de programmation. La ruée vers le marché qui s'ensuivit produisit d'ailleurs quelques bizarreries intéressantes (disons, pour être plus réaliste, des erreurs) dans la conception même du langage. Le résultat fut, et est toujours, une sorte d'espéranto qui semble étonnamment familier (mais c'est une impression trompeuse) aux développeurs déjà familiarisés avec d'autres langages de programmation.

Appelez-moi Mocha

Le nom original de JavaScript était Mocha. La première version bêta du navigateur de Netscape le rebaptisa LiveScript, et son nom de baptême définitif, donc JavaScript, apparut dans le navigateur Netscape 2, en 1995. Très rapidement, Microsoft démonta tout cela et il en introduisit un clone exact dans Internet Explorer, en le nommant Jscript pour échapper à de basses questions matérielles de droits.

Netscape soumit JavaScript à une organisation internationale de standardisation, Ecma International, et il fut adapté et standardisé sous le nom ECMAScript en 1997.



Brandon Eich, le créateur de JavaScript, n'apprécia pas cette dénomination, et le fit savoir vertement en proclamant que ECMAScript était un « nom de marque indésirable qui fait penser à une maladie de peau ».



Non seulement ECMAScript porte un nom très laid pour un langage de programmation, mais le choix par Netscape de JavaScript n'était pas non plus particulièrement heureux. Si vous savez déjà programmer en Java, ou même si vous en avez simplement entendu parler, sachez que, même si les deux langages ont certaines similitudes, il s'agit

en fait d'animaux assez différents.

Nous voulons plus d'effets !

Lorsque JavaScript fit ses débuts, il devint rapidement très populaire en tant qu'outil permettant de rendre les pages Web plus dynamiques. L'une des conséquences de l'incorporation de JavaScript dans les navigateurs Web s'appelait DHTML (pour *Dynamic HTML*). Avec le DHTML, il fut possible de produire toutes sortes d'effets amusants, comme les flocons de neige illustrés sur la Figure 1.2, mais aussi d'afficher des fenêtres pop-up, ou encore des coins de page qui se replient, et surtout des choses plus utiles comme des menus déroulants ou des validations de formulaires.

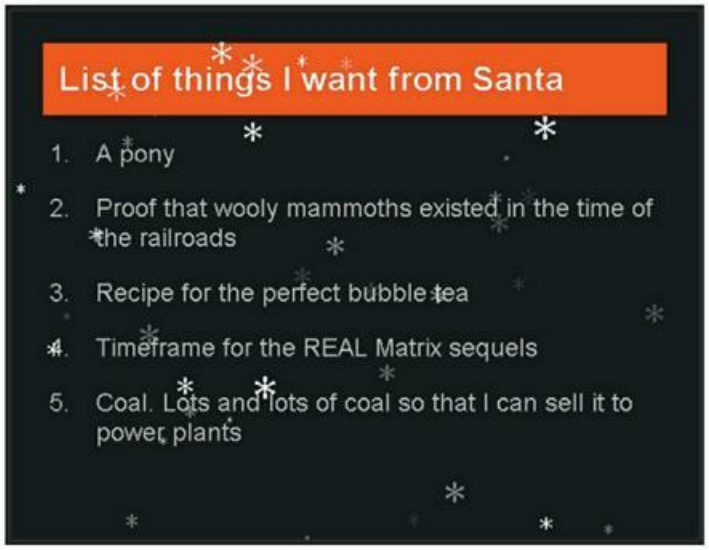

List of things I want from Santa
1. A pony
2. Proof that wooly mammoths existed in the time of the railroads
3. Recipe for the perfect bubble tea
4. Timeframe for the REAL Matrix sequels
5. Coal. Lots and lots of coal so that I can sell it to power plants

FIGURE 1.2 : Des flocons JavaScript tombant sur une page Web.

JavaScript, un langage qui se développe

JavaScript entre maintenant dans sa troisième décade. Au fil du temps, il est devenu le langage de programmation le plus utilisé au monde, et virtuellement chaque ordinateur personnel de la planète contient au moins un navigateur Web capable d'exécuter du code JavaScript.

JavaScript est suffisamment souple pour pouvoir être abordé et appris par des non-programmeurs, mais il est aussi suffisamment puissant pour permettre aux développeurs professionnels de tirer le maximum de ses capacités afin de créer des sites Web sur l'Internet, qu'il s'agisse de simples pages ou de sites gigantesques comme ceux de Google, d'Amazon, Facebook et tant d'autres !

QUELQUES ERREURS COURANTES À PROPOS DE JAVASCRIPT

Il s'est de tout temps dit des choses assez vilaines et fausses sur JavaScript. Même si certaines rumeurs ne sont pas inintéressantes, cela ne signifie pas pour autant qu'elles soient toujours vraies. Voici donc quelques erreurs courantes concernant JavaScript :

- **Mythe :** JavaScript n'est pas un vrai langage de programmation. **Réalité :** JavaScript est souvent utilisé pour réaliser des tâches triviales dans les navigateurs Web, mais cela ne diminue en rien son statut de langage de programmation. JavaScript possède de multiples fonctionnalités avancées qui ont haussé le niveau de la programmation en général, et qui sont maintenant imitées dans d'autres langages, comme PHP, C++ et même Java.
- **Mythe :** JavaScript est le fils de Java. **Réalité :** Absolument pas. Le nom JavaScript a été inventé uniquement pour des raisons marketing, car Java était extrêmement populaire lors de la sortie de JavaScript. Aujourd'hui, c'est plutôt l'inverse qui se produit.
- **Mythe :** JavaScript est nouveau. **Réalité :** JavaScript existe depuis plus de vingt ans ! Certains professionnels de la programmation en JavaScript n'étaient même pas nés lors de sa création…
- **Mythe :** JavaScript est bogué et se comporte de manière différente selon le navigateur utilisé. **Réalité :** Même si cela s'est avéré autrefois dans certains cas, les concepteurs de navigateurs Web se sont depuis longtemps mis d'accord pour prendre en charge la version standardisée de JavaScript. Par conséquent, tous les navigateurs actuels gèrent le code JavaScript de la même façon.

Un langage de script dynamique

JavaScript est souvent décrit comme étant un *langage de script dynamique*. Pour comprendre ce que cela signifie, il faut d'abord définir quelques termes et fournir certains éléments de contexte.

Les *programmes informatiques* sont des séries d'instructions qui font exécuter certaines actions par un ordinateur. Tout langage de programmation possède à cet effet un jeu d'instructions, et une certaine méthode qui permet à des êtres humains de lire ces instructions. L'ordinateur, de son côté, n'est pas capable de comprendre directement ces instructions. Il a donc besoin d'un traducteur qui transforme ces éléments qu'un programmeur normalement constitué est capable de lire (et même d'écrire) en quelque chose qui lui est familier, et que l'on appelle le *langage machine*. Selon la manière dont cette traduction est réalisée, on peut classer grossièrement les langages de programmation en deux catégories : ceux qui sont *compilés* et ceux qui sont *interprétés* (voir la Figure 1.3).

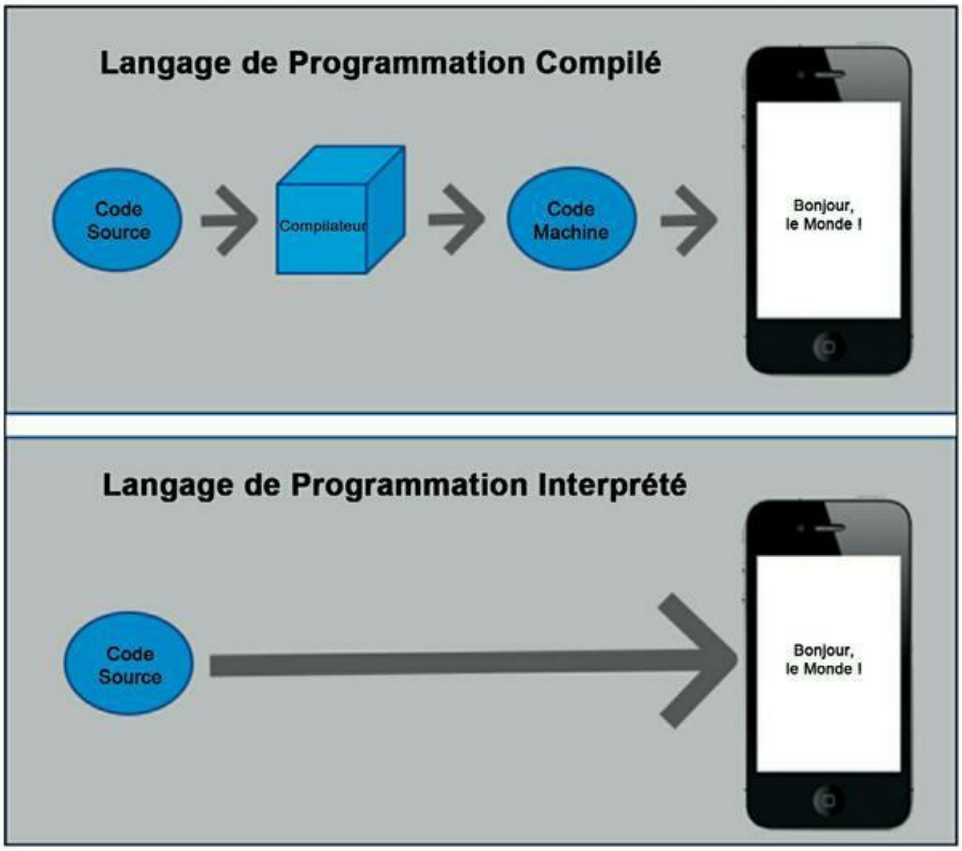


FIGURE 1.3 : Les langages de programmation sont classés en fonction du moment où la compilation est réalisée.

Langages de programmation compilés

Les *langages de programmation compilés* se définissent par le fait qu'un développeur écrit le code, puis l'envoie à un programme spécial appelé un *compilateur* qui interprète ce code et le convertit en langage machine. L'ordinateur peut ensuite exécuter le programme ainsi compilé.

Les langages de programmation compilés les plus connus sont C, C++, Fortran Java, Objective-C ou encore COBOL.

Langages de programmation interprétés

Les *langages de programmation interprétés* sont, d'un point de vue technique, compilés par l'ordinateur pour les convertir en langage machine. Mais ce processus ne se produit qu'au moment où le navigateur Web (dans le cas de JavaScript) doit exécuter le code. Les développeurs qui écrivent des programmes en langage interprété n'ont pas à se soucier des questions de compilation, puisque leur code ne sera converti qu'au moment de son exécution.

L'intérêt d'un langage interprété, c'est qu'il est possible et facile de modifier le programme à tout moment. L'inconvénient, par contre, est que la phase de compilation lors de l'exécution ajoute une étape au processus, ce qui peut évidemment nuire aux performances du programme.

Du fait de ce facteur de performances, entre autres, les langages interprétés ont souvent eu la réputation d'être moins « sérieux » que les langages de programmation compilés. Cependant, du fait que les compilateurs « à la volée » sont devenus très performants, et les processeurs des ordinateurs de plus en plus rapides, cette perception a énormément évolué. Et JavaScript a eu à cet égard un grand impact.

Parmi les langages de programmation interprétés, il convient de citer PHP, Perl, Haskell, Ruby et, bien entendu, JavaScript.

Il fait quoi, JavaScript, en réalité ?

Si vous utilisez le Web, vous vous servez tout le temps de JavaScript. La liste de choses que JavaScript est capable de réaliser est immense, en allant par exemple du simple rappel qui vous est envoyé lorsque vous oubliez de remplir un champ obligatoire dans un formulaire, jusqu'à des « monstres informatiques » tels que Google Docs ou Facebook. Voici une brève liste des applications les plus courantes de JavaScript sur le Web :

- Créer des effets chic et choc
- Valider des saisies
- Créer des effets d'enroulement
- Créer des menus déroulants ou contextuels
- Gérer le glisser et déposer
- Faire défiler des pages Web à l'infini
- Compléter automatiquement des champs
- Créer des barres de progression
- Tabuler à l'intérieur de pages Web
- Créer des listes pouvant être triées
- Proposer des zooms « magiques » (voir la Figure 1.4)

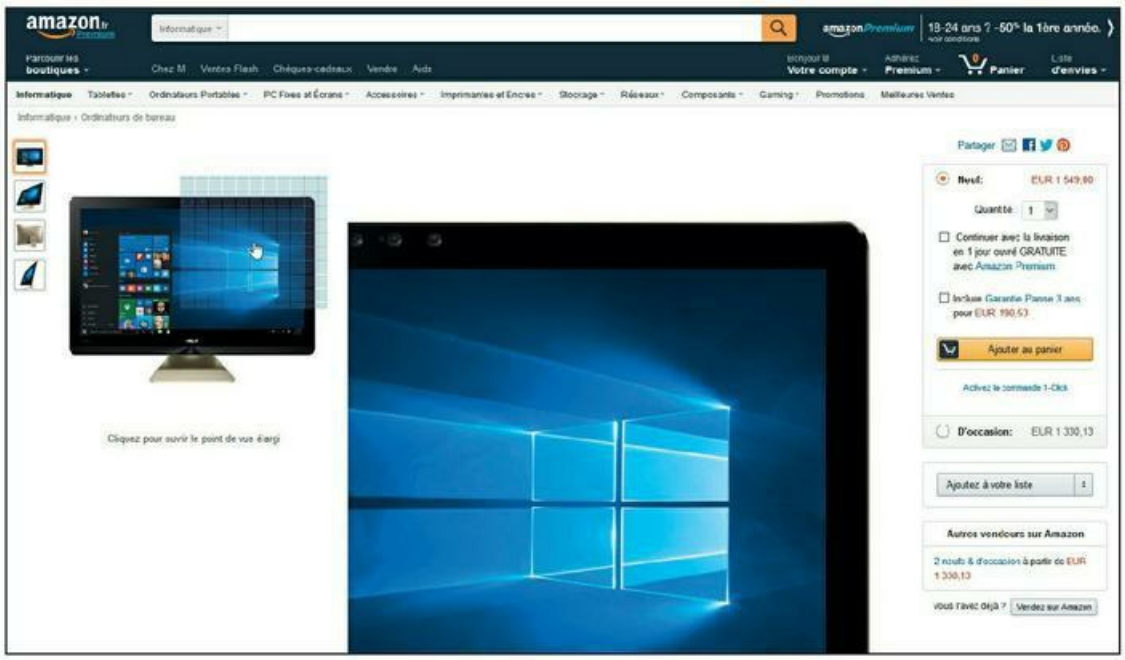


FIGURE 1.4 : Les effets de zoom dits « magiques » sont réalisés à l'aide de JavaScript.

Pourquoi JavaScript ?

JavaScript est devenu le standard pour créer des interfaces utilisateur dynamiques pour le Web. Pratiquement chaque fois que vous visitez une page Web qui contient des animations, des données évolutives, un bouton dont l'aspect change lorsque vous le survolez, ou encore un menu déroulant, vous pouvez être certain que JavaScript est entré en action. Du fait de sa puissance et de sa capacité à s'exécuter dans virtuellement n'importe quel navigateur Web, JavaScript est devenu l'outil le plus populaire et le plus nécessaire que tout développeur Web moderne doit utiliser.

JavaScript est facile à apprendre

N'oubliez pas que les langages de programmation ont été conçus pour permettre aux êtres humains que nous sommes de dialoguer avec les ordinateurs et de leur expliquer ce que nous voulons faire. Par comparaison avec le langage machine, celui que comprennent les ordinateurs, n'importe quel autre langage de programmation, qu'il soit compilé ou interprété, est facile à apprendre et à comprendre. Pour vous donner une idée des instructions auxquelles votre ordinateur est capable d'obéir, voici un exemple de code en langage machine chargé simplement d'écrire les mots « Hello, World » (voir la Figure 1.5).



FIGURE 1.5 : Tu dis quoi, Machine ?

Regardez maintenant comment vous pouvez accomplir cette tâche toute simple en JavaScript :

```
alert("Hello, World") ;
```

C'est bien plus facile, non ?



Une fois que vous avez appris les règles de base du code de la route (ce que l'on appelle la *syntaxe*), par exemple quand utiliser des parenthèses et quand utiliser des accolades ({}), JavaScript finit par ressembler à peu près à de l'anglais courant.



La première étape de l'apprentissage d'un langage, et donc aussi d'un langage informatique, consiste à surmonter la crainte de se lancer. Avec JavaScript, inutile d'avoir peur. Il existe sur le Web des milliers d'exemples de code JavaScript que tout un chacun peut récupérer et utiliser selon ses besoins. Il est en fait très facile de récupérer les outils dont vous avez besoin (voyez à ce sujet le Chapitre 2), et vous pouvez donc commencer à programmer en JavaScript en partant de petites réalisations, pour évoluer progressivement vers des créations beaucoup plus ambitieuses.

Où est JavaScript ? JavaScript est partout !

Bien que JavaScript ait été conçu à l'origine pour s'intégrer dans les navigateurs Web, il a trouvé sa place dans bien d'autres domaines. Aujourd'hui, on le trouve dans les smartphones et les tablettes, sur des serveurs Web, dans des applications du bureau, de même que dans toutes sortes de dispositifs connectés portables.

JavaScript et les navigateurs

L'endroit le plus courant pour trouver JavaScript, et là où il a été conçu à l'origine, c'est bien entendu dans les navigateurs Web. Lorsqu'il fonctionne de cette manière, on parle de JavaScript *côté client*.

Dans cette position, JavaScript ajoute de l'interactivité aux pages Web. Il accomplit cela de plusieurs manières :

- En contrôlant le navigateur lui-même et se servant des fonctionnalités de celui-ci.
- En manipulant la structure et le contenu des pages Web.

- En manipulant les styles des pages Web (comme leur mise en page ou les polices de caractères utilisées).
- En accédant à des données provenant d'autres sources.

Pour comprendre comment JavaScript est capable de gérer la structure et le style des pages Web, vous devez en savoir un peu plus sur HTML5 et sur CSS3.

HTML5

HTML (pour *Hypertexte Markup Language*, ou langage de balisage hypertexte), est le langage utilisé pour structurer les pages Web. Il travaille en se servant d'un jeu de balises afin de décomposer les contenus (textes et images) en éléments donnant aux navigateurs Web des informations sur ces contenus, et ce en définissant par exemple ce qui constitue un titre, où commence et où s'arrête un paragraphe, où une image doit être affichée, et ainsi de suite. Le code suivant vous montre un document HTML tout simple. Le résultat est illustré sur la Figure 1.6.

```
<!DOCTYPE html>
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
  <title>Hello, HTML !</title>
</head>
<body>
  <h1>Et voici HTML</h1>
  <p id="introduction">Ce simple document a été écrit en HTML (Hypertext
Markup
Language).</p>
</body>
</html>
```

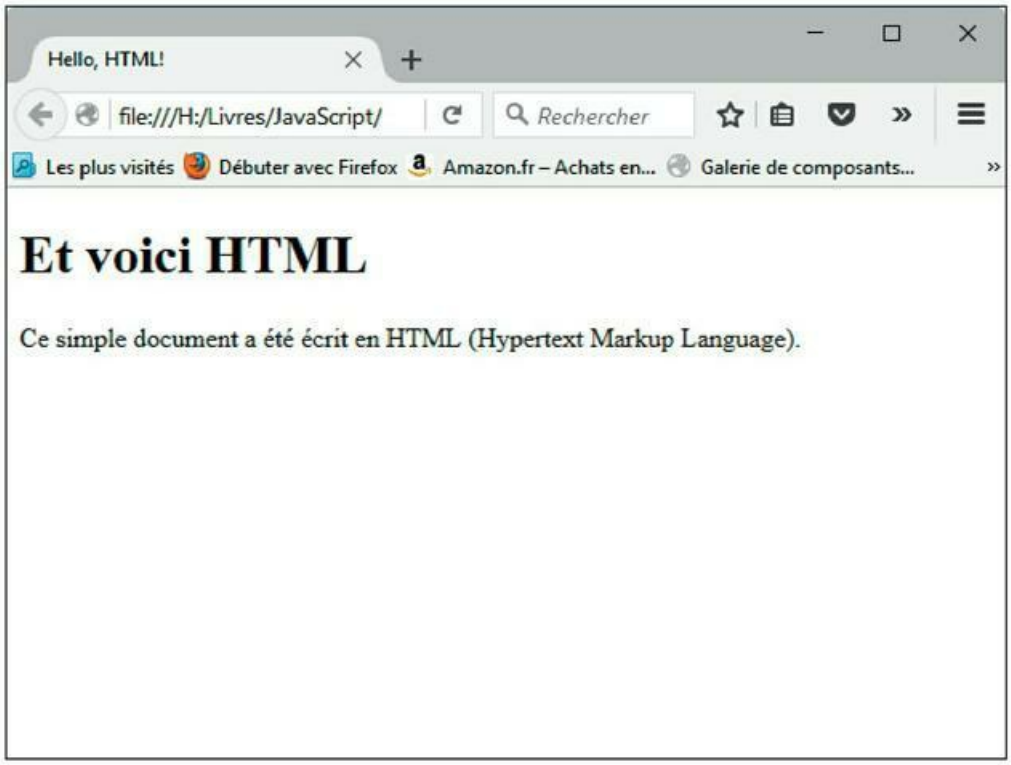


FIGURE 1.6 : Un exemple très simple de page Web écrite en HTML.



La seconde ligne, celle qui commence par la balise `meta`, sert à indiquer que le jeu de caractères français doit être utilisé par le navigateur. Sans cela, nos accents seraient remplacés par des signes bizarroïdes ! Vous retrouverez donc cette balise, qui doit être placée juste en dessous de `<html>`, dans tous les exemples de code de ce livre.

Voici tout ce que vous devez savoir pour l'instant sur HTML de manière à pouvoir vous lancer dans l'apprentissage de JavaScript :

- En HTML, les termes entourés par des crochets obliques sont appelés des *balises*.
- Une *balise de fin* (ou de terminaison), autrement dit celle qui est placée à la fin du passage ainsi marqué, se caractérise par la présence d'une barre oblique à la suite du crochet ouvrant. Exemple : </p>.
- Un groupe de deux balises (une ouvrante et une fermante), plus tout ce qui apparaît entre les deux, est appelé un *élément*.
- Les éléments sont généralement organisés d'une manière hiérarchique (autrement dit, il y a des éléments imbriqués dans d'autres éléments).
- Les éléments peuvent contenir des paires nom/valeur, appelées *attributs*. Si un élément possède des attributs, ils sont placés dans la balise ouvrante. Une paire nom/valeur assigne une valeur, placée entre guillemets ou entre apostrophes, à un nom (qui, lui, n'est pas mis entre guillemets ou apostrophes). Cette affectation est matérialisée par la présence d'un signe d'égalité. Dans cet exemple, les noms `width` (largeur) et `height` (hauteur) sont deux attributs de l'élément de structuration de page appelé `div` :
- Certains éléments ne possèdent pas de contenu, et ils n'ont donc pas besoin de balise de fin. Par exemple, la balise `img`, qui insère simplement une image dans une page Web, ressemble

à ceci :

```
<img src="monimage.jpg" width="320" height="200" alt="Ceci est une
image de mon poisson rouge">
```

Toutes les données nécessaires à l'affichage de l'image sont incluses dans cette unique balise. Il est donc inutile d'en ajouter une autre.

Lorsque vous écrivez une page en HTML, vous pouvez inclure du code JavaScript directement dans votre document, ou bien encore faire référence à du code JavaScript externe (c'est-à-dire à un fichier ayant pour suffixe `.js`). Dans les deux cas, le navigateur Web va charger le code JavaScript et l'exécuter lorsque l'utilisateur ouvre la page concernée.



JavaScript en mode *côté client* s'exécute à l'intérieur de votre navigateur Web.

CSS3

Les *feuilles de style en cascade* (ou CSS) est le langage utilisé pour ajouter des styles de formatage et de disposition aux pages Web. Le mot *style*, lorsqu'il est utilisé à propos de CSS, fait référence aux multiples aspects qui régissent la manière dont une page Web est présentée à l'utilisateur. Cela concerne notamment :

- les polices de caractères ;
- la taille des caractères ;
- les couleurs ;
- l'arrangement des éléments dans la fenêtre du navigateur ;
- la taille des éléments ;
- les bordures ;
- les arrière-plans ;
- la création de bordures aux coins arrondis autour des éléments.

Comme JavaScript, CSS peut être placé directement dans un document HTML, ou bien être stocké dans un fichier qui est lié à ce document (ce qui est de loin préférable pour la commodité de la lecture et de la compréhension). Une fois chargé, il se met immédiatement au travail et il formate le document en fonction des spécifications qu'il contient.

Les feuilles de styles CSS sont formées de *règles CSS*, qui contiennent des propriétés et des valeurs destinées à être appliquées à un élément ou à un groupe d'éléments. Par exemple :

```
p{font-size: 14px; font-color: black; font-family: Arial; sans-serif}
```

Cette règle, en la lisant de gauche à droite, spécifie que tous les éléments de type `p` (c'est-à-dire les paragraphes en HTML) devraient être affichés dans un corps de 14 pixels, en noir, et en utilisant la police Arial. Si celle-ci n'est pas présente sur l'ordinateur de l'utilisateur, le navigateur utilisera une police sans empattement du même genre.

La partie de la règle CSS qui se trouve en dehors des accolades est appelée le *sélecteur*. Il permet fort justement de sélectionner les éléments auxquels s'appliquent les propriétés spécifiées à l'intérieur de ces accolades.

Ce livre n'est pas dédié à HTML ou CSS, mais bel et bien à JavaScript. Nous fournirons donc uniquement des informations suffisantes pour pouvoir vous montrer comment HTML et CSS travaillent. Si vous voulez aller plus



loin, vous trouverez évidemment des ouvrages dédiés à ces sujets dans la vaste collection Pour les Nuls.

JavaScript est puissant !

Autrefois, JavaScript avait mauvaise réputation auprès des programmeurs du fait de sa lenteur d'exécution dans les navigateurs. De nos jours, la situation a considérablement évolué, et JavaScript fonctionne presque aussi vite que du code compilé (enfin, disons à 80 % de celui-ci). Et il progresse en permanence. Cela signifie que le JavaScript d'aujourd'hui est bien plus puissant que celui qui prévalait-il y a quelques années. Et encore beaucoup plus puissant que le JavaScript des origines, en 1995.

JavaScript est demandé !

JavaScript est non seulement le langage de programmation le plus largement connu à l'heure actuelle, mais c'est aussi l'un des plus demandés chez les professionnels. Les projections indiquent que la demande de développeurs JavaScript devrait croître de 22 % entre 2010 et 2020. L'avenir de JavaScript semble donc florissant, voire même glorieux, et c'est donc le bon moment pour l'apprendre.

Chapitre 2

Écrire votre premier programme JavaScript

DANS CE CHAPITRE :

- **Disposer votre environnement de développement**
- **Se familiariser avec le code JavaScript**
- **Comprendre le déroulement d'un programme JavaScript simple**
- **Comprendre l'importance des commentaires dans le code**

« Pour aller de l'avant, le vrai secret c'est de partir. »

Mark Twain

La création des programmes JavaScript simples n'est pas compliquée à comprendre. Dans ce chapitre, vous allez tout d'abord configurer votre ordinateur pour pouvoir travailler avec JavaScript. Vous écrirez aussi votre premier programme, et vous découvrirez la syntaxe de base qui se dissimule derrière tout ce que vous ferez à l'avenir avec JavaScript en tant que développeur.



N'oubliez pas de visiter le site Web compagnon de ce livre pour y charger les exercices qui y sont proposés. Cela vous fera gagner du temps et éviter des fautes de frappe !

Configurer votre environnement de développement

Il est important que tous les outils dont vous avez besoin soient bien rangés et configurés avant d'écrire votre premier programme JavaScript. Nous allons donc commencer par rechercher, charger et installer les outils de développement dont nous allons avoir besoin dans tout ce qui suit. Nous allons vous proposer nos propres choix, qui nous donnent parfaitement satisfaction, mais vous pouvez parfaitement avoir d'autres préférences. Nous vous proposerons d'ailleurs des alternatives dans la suite de ce chapitre. Cependant, nous vous conseillons dans tous les cas de lire cette section afin de comprendre pourquoi nous avons fait ces choix, et donc d'être mieux à même de faire les vôtres.

Une fois chacun de ces outils installés, nous dévoilerons quelques trucs et astuces qui devraient vous permettre de tirer le meilleur parti de chacun d'eux.

Télécharger et installer Google Chrome

Google Chrome est le navigateur qui a notre préférence pour travailler avec JavaScript. Bien entendu, si vous êtes habitué à jongler chaque jour avec Firefox, Safari, Opera, voire Edge (Internet Explorer étant définitivement à éviter), c'est parfait. Tous les navigateurs exécutent le code JavaScript très vite et tout à fait correctement. Cependant, certaines des instructions de ce livre sont spécifiques à Google Chrome, et c'est pourquoi nous vous recommandons dès maintenant de l'installer sur votre ordinateur. En fait, Google Chrome offre d'excellents outils qui facilitent le travail des programmeurs JavaScript, et il est à l'heure actuelle le navigateur le plus utilisé sur l'Internet.

Si Chrome n'est pas installé sur votre ordinateur, suivez donc ces étapes :

1. **Ouvrez votre navigateur habituel, et rendez-vous sur la page de Chrome (`www.google.com/chrome`).**

   Comme l'illustre la Figure 2.1, Google vous simplifie la vie au maximum.

2. **Vérifiez que la version proposée par défaut est bien celle qui correspond à votre système. Cliquez ensuite sur le bouton Télécharger Google Chrome.**

3. **Une fois le chargement terminé, procédez comme pour n'importe quelle autre application. Autrement dit, lancez le fichier servant à installer Chrome et laissez-vous guider.**

   Rien de sorcier ni de bien terrible là-dessous.



FIGURE 2.1 : Installer Chrome est facile, aussi bien sous Windows que sur Mac.

VOUS AVEZ MAINTENANT UN SUPER MOTEUR JAVASCRIPT !

Google Chrome utilise le moteur JavaScript V8 de Google pour analyser, compiler et exécuter le code JavaScript. Selon les tests de rapidité auxquels vous croyez (ou non), Chrome est ou bien le moyen le plus rapide d'exécuter des programmes JavaScript dans un navigateur, ou du moins l'un des plus véloces. La majorité des sociétés qui développent des navigateurs Web sont sans cesse en compétition entre elles. Savoir qui est le plus rapide à l'instant T n'est pas si important. En réalité, pratiquement tous les navigateurs progressent en permanence (ou presque) et chaque nouvelle version est meilleure que la précédente.

Si vous voulez voir une comparaison portant sur des tests d'exécution de JavaScript dans différents navigateurs Web, vous pouvez jeter un coup d'œil à l'adresse http://arewefastyet.com. Ce site est géré par Mozilla, le créateur du navigateur Firefox. Il contrôle et affiche en permanence les performances JavaScript des navigateurs les plus populaires. Il est actualisé plusieurs fois par jour.

Télécharger et installer un éditeur de code

Un *éditeur de code source*, comme on l'appelle généralement, est un éditeur de texte qui possède des fonctionnalités spécifiquement dédiées à la programmation. Cet outil vous aide à écrire et éditer le code de vos programmes. Celui que nous allons utiliser ici s'appelle Sublime Text.



Il existe de nombreux éditeurs de code. Si vous en avez déjà installé un qui vous convient parfaitement, utilisez-le ! Choisir un éditeur de texte avec lequel on se sent bien est une décision très personnelle, et chacun peut avoir son propre avis sur le sujet. Si vous pensez que Sublime Text n'est pas votre compagnon de demain, voyez si le Tableau 2.1 vous permet de trouver votre bonheur.

Tableau 2.1 : Quelques éditeurs de code.

Nom	Adresse Web	Compatible avec
Coda	http://panic.com/coda	Mac uniquement
Aptana	www.aptana.com	Mac ou Windows
Komodo Edit	www.activestate.com/komodo-edit/downloads	Mac ou Windows
Dreamweaver	http://adobe.com/products/dreamweaver.html	Mac ou Windows
Eclipse	www.eclipse.org	Mac ou Windows
Notepad + +	http://notepad-plus-plus.org	Windows uniquement
TextMate	http://macromates.com	Mac uniquement
BBEdit	www.barebones.com/products/bbedit	Mac uniquement
EMacs	www.gnu.org/software/emacs	Mac ou Windows
TextPad	www.textpad.com	Windows uniquement
Vim	www.vim.org	Mac ou Windows

NdT : Le choix des auteurs, Sublime Text, est très bon. Cependant, il a le défaut de ne pas être localisé en français. En programmation, ce n'est sans doute pas si grave, puisque, après tout, les langages de programmation sont basés sur une syntaxe anglophone… Parmi les éditeurs de code gratuits proposés dans le Tableau 2.1, il convient tout de même de mettre en exergue le puissant Eclipse, ou encore le plus que suffisant Notepad + +, qui dispose d'une interface en français et du support d'un très grand nombre de langages de programmation.

Dans ce livre, nous utiliserons Sublime Text, car il est populaire auprès des programmeurs JavaScript, et qu'il propose une interface utilisateur simple, tout en acceptant l'intégration de nombreux modules complémentaires permettant de pousser plus loin son expérience et de gérer des tâches complexes (voir la Figure 2.2).

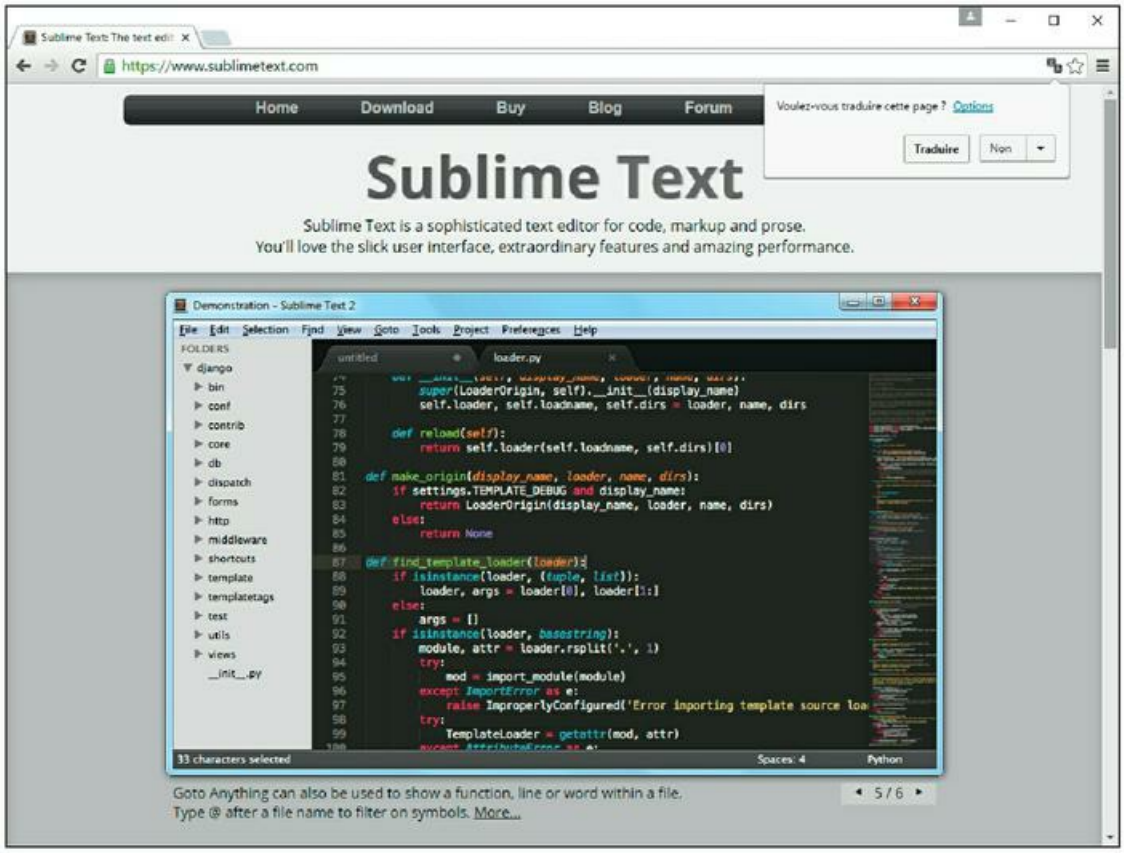

https://www.sublimetext.com
Home
Download
Buy
Blog
Forum
Sublime Text
Sublime Text is a sophisticated text editor for code, markup and prose.
You'll love the slick user interface, extraordinary features and amazing performance.
Goto Anything can also be used to show a function, line or word within a file.
Type @ after a file name to filter on symbols. More...

FIGURE 2.2 : Sublime Text est un éditeur de code qui sait rester simple tout en offrant de nombreuses fonctionnalités.

Pour installer Sublime Text, suivrez ces étapes :

1. **Ouvrez la page https://www.sublimetext.com/. Cliquez sur le bouton Download, puis choisissez la version appropriée à votre système d'exploitation.**
2. **Une fois le fichier téléchargé, ouvrez-le puis suivez les instructions d'installation.**

   C'est aussi simple que cela.

Il existe aussi des versions dites portables de Sublime Text. Dans ce cas, il n'y a pas de procédure d'installation : il vous suffit de copier le contenu du fichier compressé dans un dossier de votre choix.

Débuter avec Sublime Text

Lorsque vous lancez Sublime Text pour la première fois, vous vous trouvez face à une interface très simple. Pour l'essentiel, elle présente un curseur clignotant sur un fond sombre, ainsi qu'une barre de menus et une barre d'état (voir la Figure 2.3).

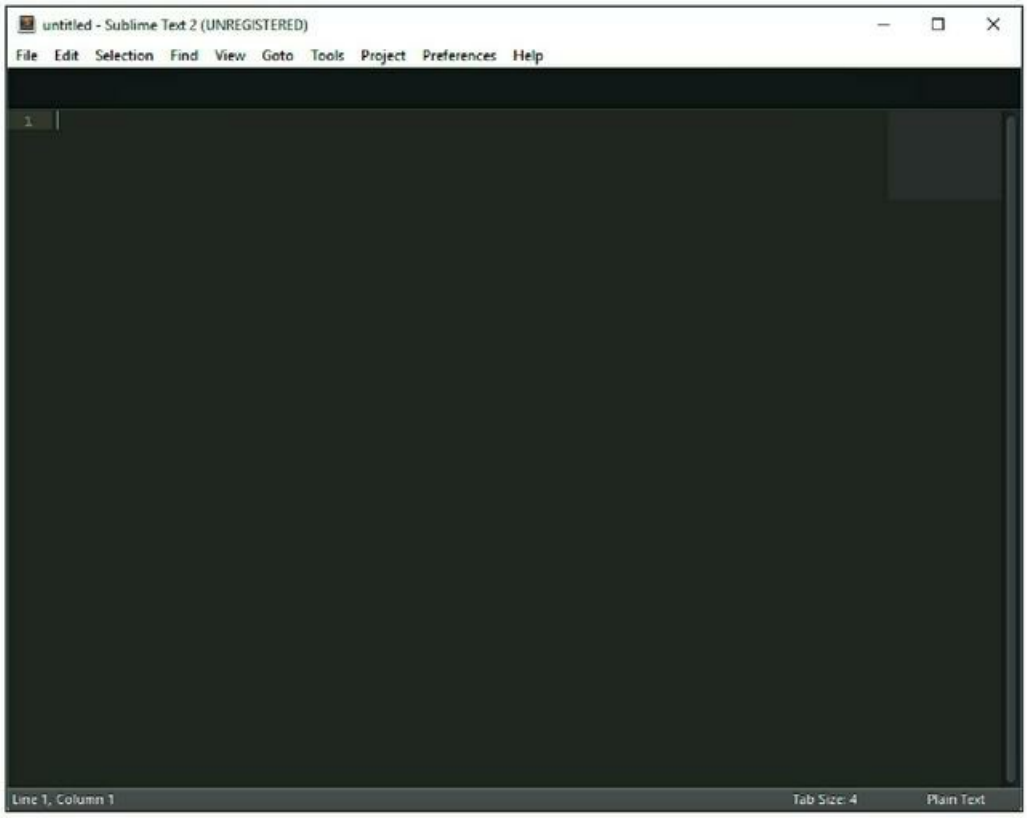

untitled - Sublime Text 2 (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
1
Line 1, Column 1
Tab Size: 4
Plain Text

FIGURE 2.3 : L'interface initiale de Sublime Text est on ne peut plus simple.

Si vous avez déjà utilisé Sublime Text, il est possible qu'un volet montrant vos fichiers ouverts, ainsi que les fichiers relevant de votre projet, soit présent à gauche de la fenêtre. Comme ce volet est utile, nous vous recommandons de l'activer.



Pour ouvrir le volet de gauche, cliquez sur le menu View (affichage), puis sur Side Bar (barre ou volet de côté, si vous préférez), et enfin sur Show Side Bar. Le résultat est illustré sur la Figure 2.4.

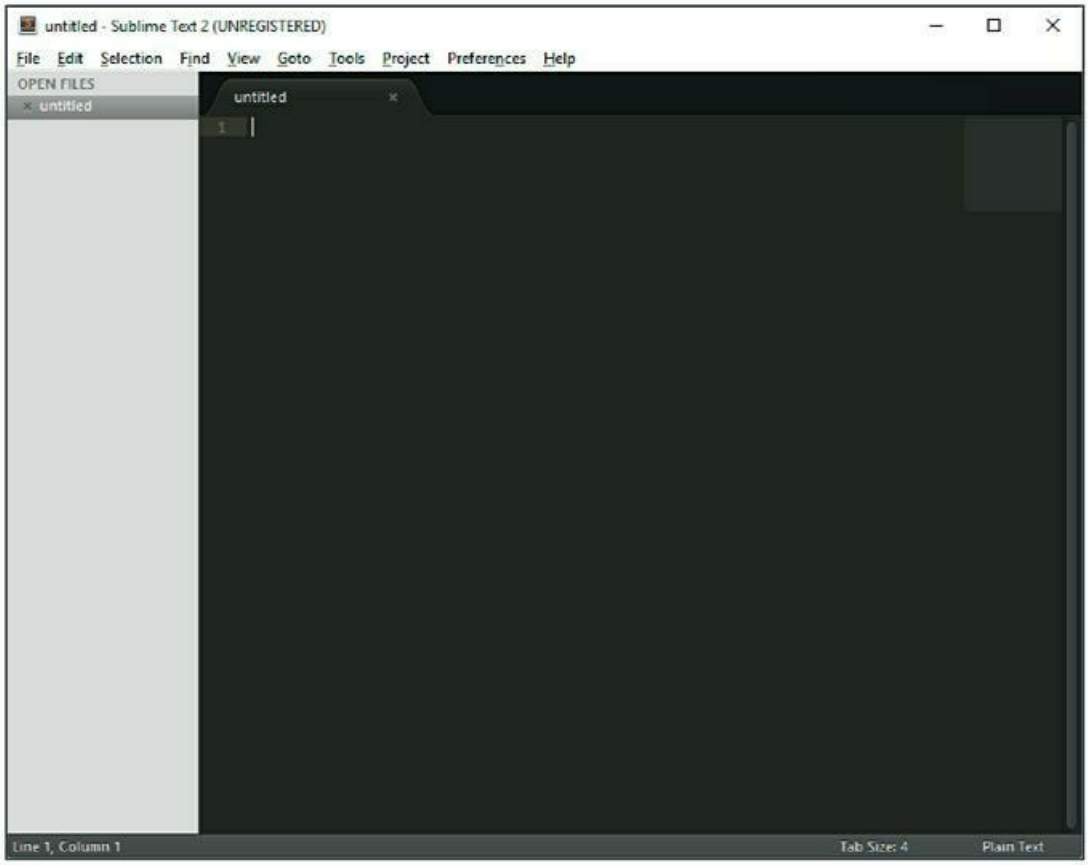


FIGURE 2.4 : La fenêtre de Sublime Text une fois le volet latéral ouvert.

Pour débuter avec un premier projet dans Sublime Text, suivez ces étapes :

1. **Choisissez dans le menu File (fichier) la commande Save (enregistrer sous).**

   La boîte de dialogue Enregistrer sous apparaît. Elle affiche le dossier de sauvegarde par défaut. Si la proposition (du genre Documents ou Mes Documents) vous convient, passez à l'Étape 2. Sinon, naviguez vers un autre emplacement sur votre ordinateur pour choisir votre dossier de destination.

2. **Créez un nouveau dossier en lui donnant un nom à votre convenance.**

3. **Tapez un nom pour votre fichier (en lui ajoutant manuellement l'extension `.html`), puis cliquez sur le bouton Enregistrer.**

   Le nom du nouveau document va apparaître dans le volet de gauche, et un onglet vient s'ajouter en haut de la fenêtre pour que vous puissiez commencer à saisir votre nouveau projet.

4. **Choisissez maintenant la commande Save Project As dans le menu Project. Sauvegardez le projet dans le dossier que vous venez de créer.**

   Les projets Sublime Text servent à mémoriser des informations sur les fichiers et les dossiers qui leur sont associés. Cela vous permet de mieux organiser tous les types de fichiers qui constituent le programme sur lequel vous travaillez.

5. **Dans le menu Project, choisissez maintenant Add Folder to Project. Sélectionnez le dossier que vous venez de créer lors de l'Étape 1, puis cliquez sur Ouvrir.**

Une nouvelle liste devrait apparaître dans le volet de gauche. Elle porte le nom de votre dossier, et contient les fichiers que vous venez de créer lors des étapes précédentes. C'est ce qu'illustre la Figure 2.5.

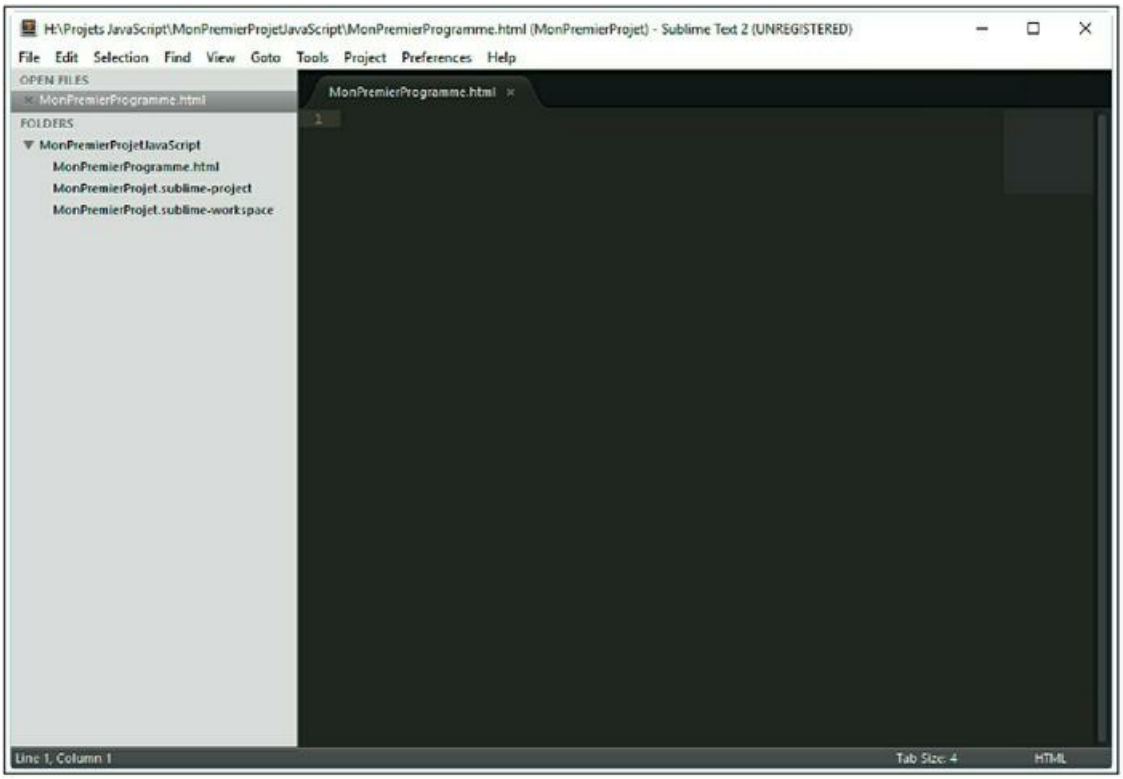


FIGURE 2.5 : Votre premier projet Sublime Text est prêt à démarrer !



Si vous ne voyez pas s'afficher le dossier du projet dans le volet de gauche, ouvrez le menu File, sélectionnez la commande Open Folder, et sélectionnez simplement le nom du dossier créé lors de l'Étape 2.



Pour bien organiser vos dossiers et vos fichiers, vous devez définir une procédure cohérente pour nommer vos documents. Par exemple, vous pourriez appeler le dossier défini lors de l'Étape 2 `MonPremierProjetJavaScript`, le fichier enregistré lors de l'Étape 3 `MonPremierProgramme`, et le projet de l'Étape 4 `MonPremierProjet`.

Choisir un schéma de coloration syntaxique

La manière dont Sublime Text met en couleur les éléments de syntaxe du langage dépend du type de code que vous écrivez, ainsi que de l'extension des fichiers. Insérez dans le fichier que vous venez de créer le code illustré sur le Listing 2.1 (ou ouvrez tout simplement le fichier listing2-1.html téléchargé depuis le site Web compagnon du livre).



Le code doit être saisi tel quel. JavaScript veut que ses mots-clés soient écrits sans fautes, et que la capitalisation (les majuscules et les minuscules) soit parfaitement respectée. À défaut de quoi, votre script pourrait ne pas donner les bons résultats, voire refuser totalement de fonctionner.

LISTING 2.1 : Exemple de fichier HTML contenant du code JavaScript.

```
<!DOCTYPE html>
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
  <title>Bonjour, HTML !</title>
  <script>
    function countToTen(){
      var count = 0;
      while (count < 10) {
         count++;
        document.getElementById("theCount").innerHTML += count + "<br>";
      }
    }
  </script>
</head>
<body onload="countToTen();">
  <h1>Apprenons à compter jusqu'à 10 avec JavaScript!</h1>
  <p id="theCount"></p>
</body>
</html>
```

La Figure 2.6 montre la manière dont Sublime Text affiche ce code.



Si vous n'aimez pas la coloration par défaut utilisée par Sublime Text pour mettre en évidence la syntaxe du langage, vous pouvez parfaitement en changer. Pour cela, ouvrez le menu Preferences et choisissez la commande Color Scheme. Parcourez la liste des thèmes proposés jusqu'à ce que vous trouviez ce qui vous convient le mieux. Dans la suite de ce livre, nous utiliserons le thème Eiffel, mieux adapté à une impression en noir et blanc que le thème par défaut (Monokai), mais aussi moins gourmand en encrage…

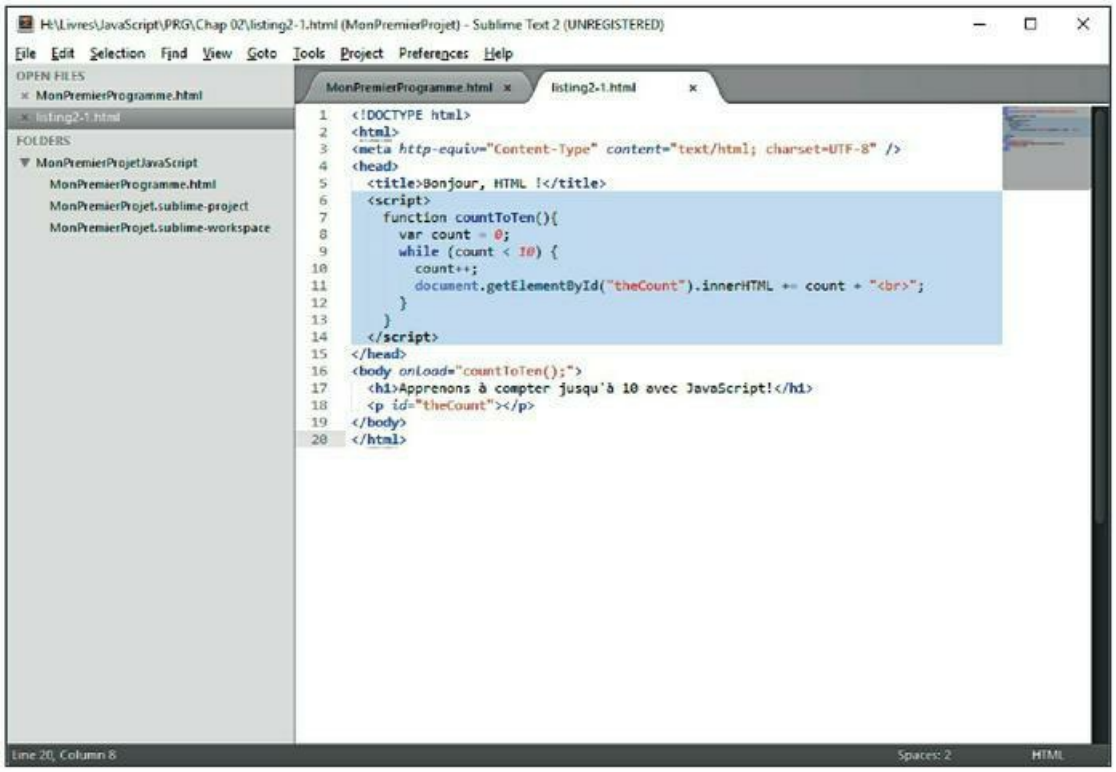


FIGURE 2.6 : Le code est affiché dans Sublime Text (ici avec le thème Eiffel).

Si vous voulez tout de suite essayer ce premier programme, suivez ces étapes :

1. **Sauvegardez votre fichier (File puis Save As).**
2. **Ouvrez votre navigateur Chrome, puis appuyez sur Ctrl + O.**

   Une fenêtre d'ouverture de fichier apparaît.
3. **Naviguez jusqu'au dossier qui contient votre programme et sélectionnez-le.**
4. **Cliquez sur le bouton Ouvrir.**

   Le fichier s'ouvre dans le navigateur.

Le résultat est illustré sur la Figure 2.7. Si ce n'est pas ce que vous obtenez, vérifiez très soigneusement votre code. Il est tout à fait possible que vous ayez fait une ou deux fautes de frappe. Et n'oubliez pas de sauvegarder votre fichier après chaque modification !



FIGURE 2.7 : Un petit programme de comptage affiché dans Chrome.

Raccourcis Sublime Text utiles

En apparence, Sublime Text est un éditeur qui semble ordinaire, mais ne vous y fiez pas ! La véritable marque d'un programmeur de haut niveau est sa capacité à utiliser des raccourcis clavier pour accélérer la saisie du code et opérer les modifications de celui-ci aussi vite que possible. Le Tableau 2.2 liste quelques-uns des multiples raccourcis clavier fournis par Sublime Text. Exercez-vous à mémoriser ceux qui vous paraissent les plus efficaces pour votre travail, et vous impressionnerez très vite vos amis et vos collègues par vos super-compétences.

Tableau 2.2 : Les principaux raccourcis clavier de Sublime Text.

Mac	Windows	Description
Command + X	Ctrl + X	Supprimer une ligne
Command + Retour	Ctrl + Entrée	Insérer une ligne après
Command + Maj + Retour	Ctrl + Maj + Entrée	Insérer une ligne avant
Command + Control + Flèche haut	Ctrl + Maj + Flèche haut	Déplacer la ligne ou la sélection vers le haut
Command + Control + Flèche bas	Ctrl + Maj + Flèche bas	Déplacer la ligne ou la sélection vers le bas
Command + L	Ctrl + L	Sélectionner une ligne, répéter pour continuer la sélection
Command + D	Ctrl + D	Sélectionner une ligne, répéter pour continuer la sélection
Control + M	Ctrl + M	Atteindre une parenthèse fermante, répéter pour passer à la suivante

Control + Maj + M	Ctrl + Maj + M	Sélectionner tout le contenu des parenthèses courantes
Command + K + Command + K	Ctrl + k + k	Supprimer le code depuis la position courante du curseur jusqu'à la fin de la ligne
Command + K + Suppr	Ctrl + K + Suppr	Supprimer le code depuis la position courante du curseur jusqu'au début de la ligne
Command +]	Ctrl +]	Indenter la ou les lignes courantes
Command + [	Ctrl + [	Supprimer l'indentation de la ou des lignes courantes
Command + Maj + D	Ctrl + Maj + D	Dupliquer une ou plusieurs lignes
Command + J	Ctrl + J	Joindre la ligne suivante à la fin de la ligne courante
Command + /	Ctrl + /	Commenter la ligne courante, ou supprimer le commentaire
Command + Option + /	Ctrl + Maj + /	Commenter la sélection courante
Command + Y	Ctrl + Y	Répéter la commande exécutée par le dernier raccourci clavier
Command + Maj + V	Ctrl + Maj + V	Coller et indenter correctement
Control + Espace	Ctrl + Espace	Sélectionner la sélection auto-complétée suivante
Control + Z	Ctrl + Z	Annuler la dernière modification
Control + U	Ctrl + U	Sélectionner la dernière modification sans l'annuler, ou atteindre la dernière modification avant de l'annuler en répétant le raccourci
Control + Maj + Haut	Ctrl + Alt + Haut	Sélectionner une colonne vers le haut
Control + Maj + Bas	Ctrl + Alt + Bas	Sélectionner une colonne vers le bas
Control + Maj + W	Alt + Maj + W	Transformer la sélection en paragraphe

Lire le code JavaScript

Avant de commencer à écrire des programmes JavaScript, vous devez connaître quelques règles de base sur ce langage :

» **JavaScript est sensible à la casse.** En d'autres termes, JavaScript fait la différence entre les majuscules et les minuscules. C'est là un point essentiel sur lequel nous aurons à insister à

plusieurs reprises dans ce livre. C'est en effet une des principales causes d'erreur dans le code JavaScript. De son point de vue, les mots *programme* et *Programme* sont totalement différents.

- **JavaScript n'est pas sensible aux espaces blancs.** La notion d'*espace blanc* concerne aussi bien les espaces proprement dites que les tabulations et les sauts de ligne. Dans du code JavaScript, vous pouvez parfaitement insérer une espace, plusieurs espaces, une tabulation ou même un saut de ligne (du moins dans la plupart des cas). JavaScript les ignorera. La seule exception concerne les textes que vous voulez faire afficher à l'écran. Dans ce cas, évidemment, ces espaces blancs apparaîtront dans la fenêtre du navigateur. Le mieux à faire, c'est d'insérer suffisamment d'espaces blancs dans votre code pour que celui-ci soit facile à lire et de vous en tenir à une utilisation cohérente de cette méthode.
- **Faites attention aux mots réservés.** JavaScript est un langage, et donc il contient un ensemble de mots qui lui sont propres et qu'il est capable de comprendre. Nous y reviendrons dans le Chapitre 3. Pour l'instant, retenez que des mots tels que *function*, *while*, *break* ou encore *with* sont réservés, et donc inutilisables en dehors du rôle qui leur est dévolu.
- **JavaScript aime les points-virgules.** Le code JavaScript est constitué d'*instructions*. Une instruction, c'est un peu comme une phrase. Vous pouvez vous la représenter comme un bloc de base du code JavaScript, de même qu'une phrase est un bloc de base dans un paragraphe de texte. En JavaScript, les instructions se terminent par un point-virgule (de même qu'une phrase se termine par un point).



Si vous oubliez d'insérer un point-virgule à la fin d'une instruction, JavaScript le fera à votre place. Cependant, cela peut produire des résultats imprévisibles. Il est hautement préférable de prendre la bonne habitude de toujours terminer une instruction en tapant un point-virgule. C'est un réflexe à acquérir.

Exécuter JavaScript dans la fenêtre du navigateur

Bien qu'on puisse le rencontrer dans de multiples environnements, l'endroit le plus fréquent où on le voit en action, c'est dans la fenêtre d'un navigateur Web. Contrôler les entrées et les sorties, manipuler des pages Web, gérer des événements courants du navigateur comme des clics ou des défilements, ou encore mettre la main sur le navigateur lui-même, voilà autant de tâches, parmi bien d'autres, pour lesquelles JavaScript a été conçu !

Pour exécuter du code JavaScript dans un navigateur Web, vous avez trois options sur lesquelles nous allons revenir dans les pages qui suivent :

- Placer directement le code dans un attribut d'événement HTML.
- Placer le code entre des balises de début et de fin de script.
- Placer le code dans un document séparé et l'inclure dans votre document HTML.

La plupart du temps, vous combinerez ces trois techniques dans une même page Web. Cependant, savoir quand utiliser chacune de ces méthodes est important, et c'est là quelque chose que vous apprendrez à maîtriser au fur et à mesure que vous progresserez.

Utiliser JavaScript dans un attribut d'événement

HTML

HTML possède plusieurs attributs spéciaux qui sont conçus pour déclencher du code JavaScript lorsque quelque chose de particulier se passe dans la fenêtre du navigateur, ou lorsque l'utilisateur effectue une certaine action. Voici un exemple qui montre un bouton HTML possédant un attribut d'événement capable de répondre à des clics de souris :

```
<button id="bigButton" onclick="alert(Bonjour, le Monde !');">Cliquez
ici</
button>
```

Lorsque l'utilisateur clique sur le bouton créé par cet élément HTML, une fenêtre pop-up affichant le message spécifié apparaît.

HTML possède environ 70 attributs d'événements différents. Le Tableau 2.3 liste ceux qui sont le plus couramment employés.

Tableau 2.3 : Les attributs d'événements HTML les plus utilisés.

Attribut	Description
onload	Exécute le script une fois le chargement des pages terminé.
onfocus	Exécute le script lorsque l'élément concerné a le focus (par exemple lorsqu'un champ de texte est actif).
onblur	Exécute le script lorsque l'élément concerné perd le focus (par exemple lorsque l'utilisateur clique sur un nouveau champ de texte dans un formulaire).
onchange	Exécute le script lorsque la valeur d'un élément change.



Même si cette technique est simple à utiliser, elle est mal considérée par la plupart des programmeurs JavaScript. Pour autant, si nous y faisons référence dans ce livre, c'est parce qu'elle est à la fois très largement employée et qu'elle est simple à apprendre. Mais, à ce stade, n'oubliez pas qu'il existe de meilleurs moyens pour réagir à des événements que d'utiliser des attributs d'événements. Nous y reviendrons dans le Chapitre 11.

Utiliser JavaScript dans un élément de script

Les éléments HTML vous permettent d'incorporer du code JavaScript dans vos documents. Bien souvent, les éléments de script sont placés dans l'en-tête (la balise `head`). En fait, il s'agit généralement d'une obligation. Mais, pour autant, dans les navigateurs modernes, les éléments de script peuvent aussi bien être insérés dans l'en-tête que dans le corps des pages Web.

Le formatage des éléments de script est très simple :

```
<script>
 (insérez votre code JavaScript ici)
</script>
```

Vous avez rencontré un exemple de ce type de script dans le Listing 2.1. Le Listing 2.2 montre un autre exemple dans lequel un document HTML incorpore une balise contenant du code JavaScript. Dans ce cas, cependant, ce script a été placé à la fin de l'élément `body`.

LISTING 2.2 : Exemple de script JavaScript dans un document HTML.

```
<!DOCTYPE html>
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
  <title>Bonjour, HTML!</title>
</head>
<body>
  <h1>Comptons jusqu'à 10 avec JavaScript!</h1>
  <p id="theCount"></p>
  <script>
    var count = 0;
    while (count < 10) {
      count++;
      document.getElementById("theCount").innerHTML += count + "<br>";
    }
  </script>
</body>
</html>
```

Si vous créez un nouveau fichier dans Sublime Text, que vous y collez le code du Listing 2.2, puis que vous ouvrez ce code dans un navigateur Web, vous constaterez que le résultat est exactement le même que celui qui est illustré sur la Figure 2.1.

Insertion du script et exécution de JavaScript

Normalement, les navigateurs Web chargent et exécutent les scripts au cours du chargement des pages Web. Une page Web est toujours lue de haut en bas, exactement comme si vous lisiez une page de texte. Mais il peut aussi arriver que vous vouliez qu'une certaine page soit totalement chargée avant qu'un script ne soit exécuté. Dans le cas du Listing 2.1, ce résultat a été obtenu en utilisant l'attribut `onload` placé dans l'élément `body`. Une autre manière courante de décaler cette exécution consiste à déplacer le code vers la fin du document, comme dans le Listing 2.2.

Limitations de JavaScript dans les éléments <script>

Même si cette méthode est couramment utilisée et plus facilement acceptée que l'intégration dite en ligne de JavaScript (en plaçant le code dans des attributs d'événements), incorporer du code JavaScript dans un élément de script induit un certain nombre de limitations.

La limite la plus importante est que ce type d'intégration ne fonctionne que dans des pages Web où ce type de code se trouve. En d'autres termes, si vous placez votre code JavaScript dans un élément `script`, vous devrez copier et coller ce code dans chacune des pages où il doit être exécuté. Dans le cas de sites Web qui peuvent contenir des centaines et des centaines de pages, cette technique peut devenir un véritable cauchemar.

Quand convient-il d'utiliser JavaScript dans des éléments <script>

Cette méthode d'incorporation a ses propres usages. Dans le cas de petits codes qui se contentent d'appeler d'autres morceaux de code JavaScript et qui changent rarement (si ce n'est jamais), il est tout à fait acceptable, et même plus rapide, pour le chargement et l'affichage des pages Web, de faire appel à cette technique pour faire en sorte que les requêtes au serveur soient limitées au maximum.

Dans le cas de sites qui ne comportent qu'une seule page, ce type de script est également une très bonne méthode, puisque toute évolution ne sera faite que sur cette page.

Pour autant, et en tant que règle d'ordre général, vous devriez toujours tenter de minimiser la quantité de code JavaScript que vous incorporez directement dans un document HTML. La maintenance du code en sera plus facile à gérer, et l'organisation de celui-ci en sera facilitée.

Inclure des fichiers JavaScript externes

La troisième, mais aussi la plus populaire, méthode pour inclure du code JavaScript dans un document HTML consiste à utiliser l'attribut `src` dans l'élément de script.

Un élément de script contenant un attribut `src` fonctionne exactement comme si le code était inclus à l'intérieur de balises, si ce n'est que le code est chargé, *via* cet attribut `src`, à partir d'un fichier distinct. La ligne qui suit montre un exemple d'élément de script associé à un attribut `src` :

```
<script src="monScript.js"></script>
```

Ici, vous devez disposer d'un fichier appelé `monScript.js`. Dans ce cas, ce fichier doit se trouver dans le même dossier que votre document HTML. Cette méthode comporte plusieurs avantages :

- Vos fichiers sont plus propres et plus clairs.
- Vous vous facilitez l'existence, car toute modification du code JavaScript ne doit être effectuée que dans un endroit unique, qu'il s'agisse de le faire évoluer ou de corriger des erreurs.

Créer un fichier .js

Créer un fichier JavaScript externe est semblable à l'écriture d'un document HTML ou de tout autre type de fichier. Pour remplacer par exemple le code du Listing 2.1 par un fichier JavaScript externe, suivez ces étapes :

1. **Dans Sublime Text, choisissez la commande New File dans le menu File (vous créez donc un nouveau document).**
2. **Copiez tout ce qui se trouve entre les balises `<script>` et `</script>` dans le document du Listing 2.1, puis copiez-le dans votre nouveau fichier `.js`.**

   Notez que votre fichier JavaScript externe ne contient pas les éléments `<script>`.
3. **Sauvegardez votre fichier sous un nom de votre choix dans le même dossier que le document HTML `MonPremierProgramme.html`.**
4. **Dans le document MonPremierProgramme.html, modifiez votre élément `<script>` pour le remplacer par un attribut `src`, comme ceci :**

   ```
   <script src="countToTen.js"></script>
   ```

Votre copie de MonPremierProgramme.html devrait à ce stade ressembler à ceci :

```
<!DOCTYPE html>
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
  <title>Bonjour, HTML !</title>
  <script src="countToTen.js"></script>
</head>
<body onload="countToTen();">
  <h1>Apprenons à compter jusqu'à 10 avec JavaScript!</h1>
  <p id="theCount"></p>
</body>
</html>
```

Votre nouveau fichier, countToTen, se présente de la manière suivante :

```
function countToTen(){
  var count = 0;
  while (count < 10) {
    count++;
    document.getElementById("theCount").innerHTML +=
      count + "<br>";
  }
}
```

Une fois ce travail réalisé, les différents composants de votre projet vont apparaître dans le volet de gauche de Sublime Text.

Conserver vos fichiers .js bien organisés

Les fichiers JavaScript externes peuvent parfois devenir très volumineux. Dans la plupart des cas, il est judicieux de les découper en fichiers plus petits, et de les organiser selon les fonctions qu'ils contiennent. Par exemple, un fichier JavaScript va intégrer divers scripts servant à gérer la manière dont un utilisateur peut se connecter à votre programme, tandis qu'un autre prendra en charge tout ce qui concerne les fonctions de blog de votre site.

Dans le cas de programmes plus réduits, par contre, il est généralement suffisant de n'utiliser qu'un seul fichier qui recevra un nom générique, du genre `apps.js`, `main.js` ou encore `scripts.js`.

Les fichiers JavaScript n'ont pas besoin de se trouver dans le même dossier que le document HTML dans lequel ils doivent être inclus. En fait, nous vous recommandons de créer un nouveau dossier spécifique pour y enregistrer vos fichiers JavaScript externes. La plupart des programmeurs que nous connaissons donnent à ce dossier le nom `js`.

Pour créer un sous-dossier `js` depuis votre projet Sublime Text et y déplacer le fichier JavaScript, suivez ces étapes :

1. **Cliquez droit sur le nom de votre projet dans le volet de gauche de Sublime Text.**

   Un sous-menu apparaît.

2. **Dans le menu contextuel qui s'affiche, choisissez l'option New Folder (nouveau dossier).**

Sublime Text affiche en bas de sa fenêtre un bandeau vous permettant de définir le nom du nouveau dossier.

3. **Dans le bandeau, tapez le nom `js` et appuyez sur Entrée.**

   Le nom du dossier s'affiche dans le volet de gauche, sous l'intitulé du projet.

4. **Ouvrez `countToTen.js`, puis choisissez la commande Save As dans le menu File, et sauvegardez votre fichier dans votre nouveau dossier, `js`.**

5. **Cliquez droit sur la version de `countToTen.js` qui se trouve en dehors de votre dossier, puis choisissez Delete File (supprimer le fichier) dans le menu contextuel qui s'affiche.**

6. **Ouvrez maintenant `MonPremierProgramme.html` et changez le contenu de la balise <script> de manière à la faire pointer vers votre fichier js, comme ceci :**

```
<script src="js/countToTen.js"></script>
```

Lorsque vous allez exécuter `MonPremierProgramme.html` dans la fenêtre de votre navigateur (ou simplement y rafraîchir la version actuellement chargée), le résultat devrait être exactement le même qu'avant le déplacement du fichier `js`.

Utiliser la console de développement JavaScript

Il est parfois pratique de pouvoir exécuter des commandes JavaScript sans créer de page HTML contenant votre script, ou incluant un fichier de script distinct. Pour cela, vous allez utiliser dans ce qui suit la console JavaScript de Chrome (voir la Figure 2.8).

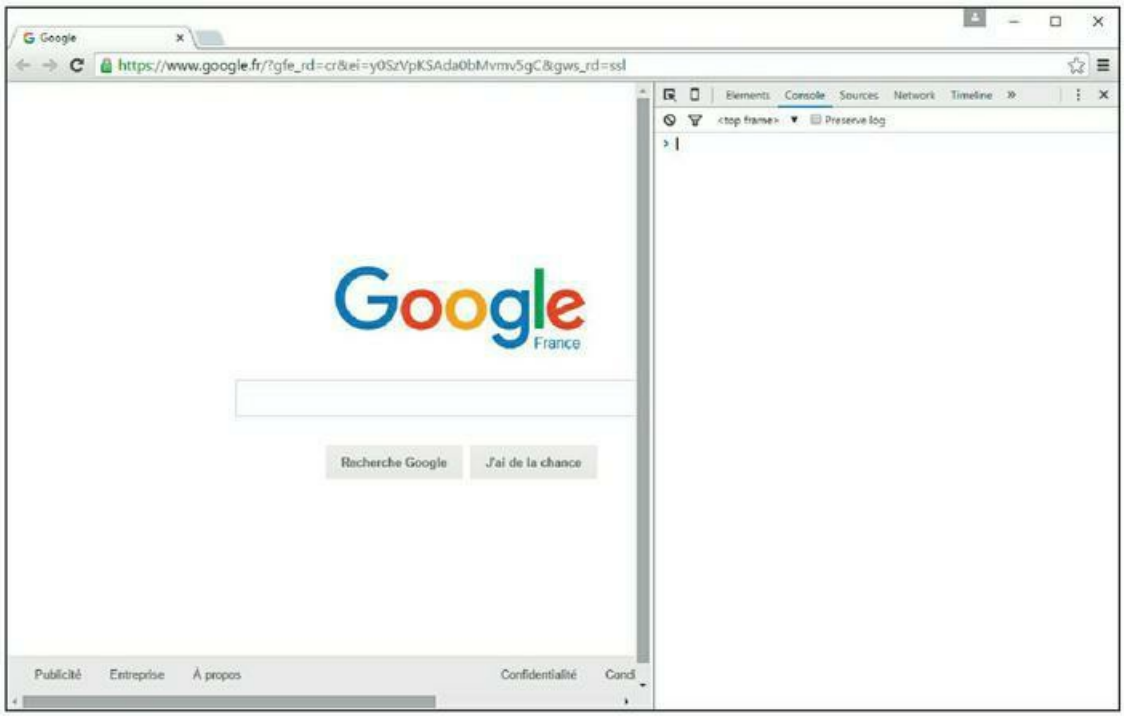


FIGURE 2.8 : Google Crome et sa console JavaScript.

Pour accéder à cette console, ouvrez Chrome si ce n'est fait, et localisez le menu qui se trouve en haut et à droite de la fenêtre du navigateur (les trois barres horizontales). Ouvrez ce menu. Cliquez sur Plus d'outils, puis sur

Outils de développement. Dans la fenêtre qui apparaît à l'intérieur de Chrome, cliquez dans le bandeau du haut sur l'option Console.



Vous pouvez aller encore plus vite en vous servant du raccourci Alt + Command + J (Mac) ou Ctrl + Maj + J (Windows).



La console de Chrome est peut-être la meilleure amie du développeur JavaScript. En premier lieu, elle vous permet de tester et d'exécuter rapidement et facilement du code JavaScript. C'est, en outre, elle aussi qui vous signale vos erreurs ; elle possède des fonctionnalités qui vous aident à localiser et à résoudre les problèmes posés par votre code.

Une fois que vous avez ouvert la console JavaScript, vous pouvez commencer à y saisir des commandes. Celles-ci sont exécutées dès que vous appuyez sur la touche Entrée. Pour faire un essai, ouvrez la console JavaScript et tapez les commandes suivantes en appuyant sur Entrée à la fin de chaque ligne :

```
1080/33
40 + 2
40 * 34
100%3
34++
34--
```

La Figure 2.9 illustre les réponses que vous obtenez.

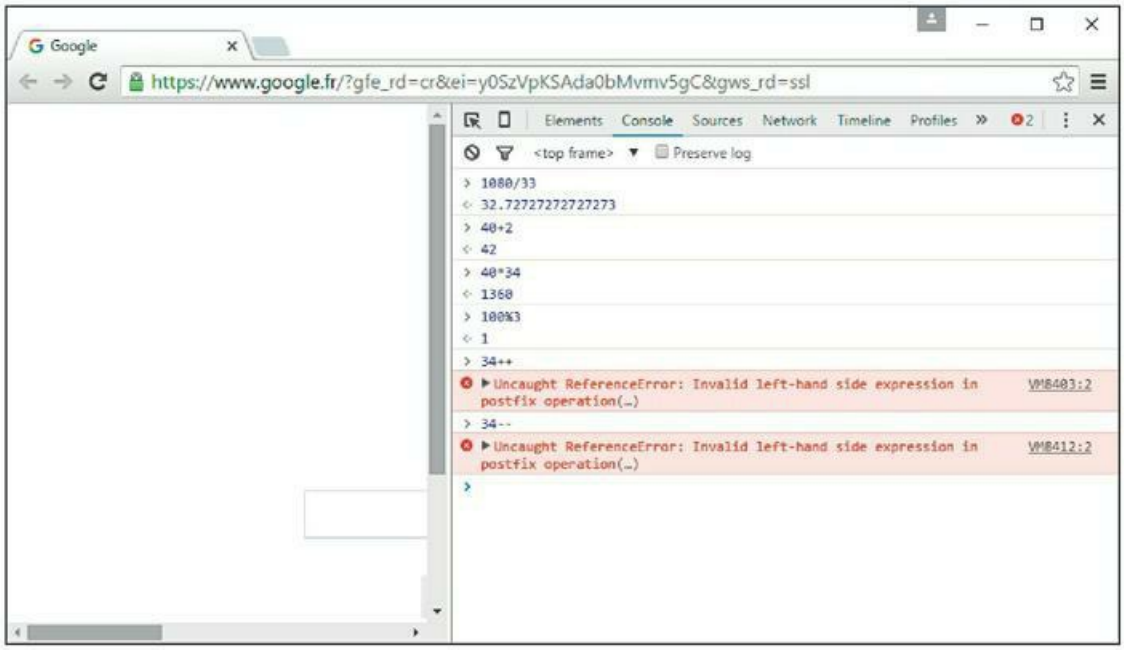


FIGURE 2.9 : La Console JavaScript de Chrome réagit instantanément à vos commandes.

Commenter votre code

Plus vous apprendrez de commandes JavaScript, plus vous écrirez des programmes conséquents, et plus vous aurez besoin de nouer des coins de mouchoirs pour vous rappeler du rôle que joue telle ou telle ligne de code, ou bien tel ou tel script. Dans la langue des programmeurs, ces rappels s'appellent des *commentaires*. Et le fait d'ajouter des commentaires dans le code est tout bonnement appelé *commenter*.



Le moteur de JavaScript ignore totalement les commentaires. Ils sont juste là pour vous et pour tous ceux qui ont besoin de lire votre code. C'est là que vous expliquez les choses, que vous les clarifiez, que vous explicitez votre

démarche, ou encore que vous notez des idées à développer dans le futur.

Commenter votre code est *toujours* une bonne idée. Même si vous pensez que ce code se comprend de lui-même au moment où vous l'écrivez, nous vous garantissons que vous allez déchanter quand vous aurez besoin d'y retravailler dans quelques mois. Ce qui vous paraissait si évident à l'époque vous semblera plus tard totalement incompréhensible. Foi de programmeur expérimenté !

JavaScript vous offre deux moyens de définir quelque chose comme étant un commentaire :

- Commentaire tenant sur une seule ligne.
- Commentaire réparti sur plusieurs lignes.

Commentaire sur une seule ligne

Un commentaire tenant sur une seule ligne débute par deux barres obliques inversées : //. Tout ce qui suit ces caractères sur la ligne courante, et seulement celle-ci, sera ignoré par JavaScript.

Ce genre de commentaire n'est pas forcément placé en début de ligne. Il est même assez courant de le placer juste après une instruction qui a besoin d'être explicitée. Par exemple :

```
pizzas = pizza + 1 // ajoute une pizza à la commande
```

Commentaires multilignes

Les commentaires répartis sur plusieurs lignes débutent par /*. Ces caractères indiquent à JavaScript qu'il doit ignorer tout ce qui suit jusqu'à ce qu'il rencontre les caractères */. Cette technique est utile lorsque le programme a besoin d'un plus long développement pour la documentation du code. Par exemple :

```
<script>
    /*
    La fonction addThing répond aux clics sur le bouton
    "Telle est ma volonté !", récupère la valeur dans le
    champ de saisie, et ajoute la nouvelle chose à la liste.
    */
    function addThing(){
      /*
      Recherche l'élément HTML ayant pour id = "thingToDo"
      et récupère sa valeur courante.
  */
  var newThing = document.getElementById("thingToDo").value;
  /*
  Trouve l'élément ayant pour id="listOfThings" et change la valeur entre
ses
  balises de début et de fin en y ajoutant le texte saisi, suivi
  d'un <br>.
  */
  document.getElementById("listOfThings").innerHTML += newThing + +  "
<br>";
}
</script>
```

Utiliser des commentaires pour éviter l'exécution de code

Les commentaires sont utiles, et même nécessaires, pour clarifier et expliquer votre code. Mais ils peuvent aussi souvent servir à isoler des parties du code de manière à rechercher de possibles problèmes. Par exemple, si nous voulions voir comment se comporterait la fonction `countToTen` si nous supprimions la ligne de la boucle qui incrémente la valeur du compteur, nous pourrions transformer cette ligne en commentaire, de cette manière :

```
function countToTen(){
  var count = 0;
  while (count < 10) {
    // count++;
    document.getElementById("theCount").innerHTML += count + "<br>";
  }
}
```

Si vous exécutez ce programme, la ligne `count++` ne sera plus exécutée, et donc le code se contentera d'afficher des zéros jusqu'à l'infini (ou du moins jusqu'à ce que vous refermiez la fenêtre du navigateur).



Le monstre ainsi créé s'appelle une *boucle infinie*. Si vous exécutez la version du code ainsi modifiée, votre ordinateur ne sera pas endommagé. En revanche, le processeur risque de tourner en rond aussi vite qu'il le peut jusqu'à ce que vous vous décidiez à refermer la fenêtre du navigateur dans lequel le programme est exécuté.

Chapitre 3

Travailler avec les variables

DANS CE CHAPITRE :

- **Créer et utiliser des variables**
- **Comprendre la portée des variables**
- **Connaître les types de données de JavaScript**
- **Nommer les variables**
- **Utiliser des fonctions intégrées pour travailler avec les variables**

« La beauté est variable, la laideur est constante. »

Douglas Horton (1891-1968)

Dans ce chapitre, vous allez découvrir comment créer des variables, comment les remplir avec des valeurs, comment utiliser des fonctions pour trouver le type des données contenues dans vos variables, comment effectuer des conversions entre différents types de données et enfin comment manipuler les données de vos variables.



N'oubliez pas de visiter le site Web compagnon de ce livre pour y télécharger les exemples de code de ce chapitre !

Comprendre les variables

Les variables sont des noms qui représentent quelque chose dans un programme. De la même manière que *x* désigne en algèbre une valeur non connue, ou même l'endroit où est caché le trésor sur une vieille carte de pirate, les variables sont utilisées dans les programmes pour représenter quelque chose d'autre.

Vous pouvez vous figurer les variables comme des conteneurs dans lesquels se trouvent des données. Vous attribuez un nom à ces conteneurs, et vous pouvez ensuite vous servir de ce nom pour récupérer le contenu du conteneur correspondant, ou encore pour y mettre autre chose.

Sans ces variables, un programme informatique ne servirait pas à grand-chose, disons uniquement à effectuer toujours le même calcul ou afficher le même texte. Prenons l'exemple suivant :

```
alert (3 + 7) ;
```

Pas de variable ici. Cette ligne de code additionne les chiffres 3 et 7, puis affiche le résultat dans le navigateur.

Si vous n'avez pas besoin d'additionner régulièrement 3 et 7 (ce qui poserait un sérieux problème quant à vos souvenirs d'école primaire, et donc aussi quant à votre capacité à programmer), ce programme n'a aucun intérêt. Grâce aux variables, vous pouvez créer un programme bien plus généraliste qui additionnera deux nombres et affichera le résultat. Par exemple :

```
var firstNumber = 3;
var secondNumber = 7;
```

```
var total = Number(firstNumber) + Number(secondNumber);
alert (total);
```

D'accord. Ici, l'ajout de variables n'apporte pas grand-chose, puisque les nombres restent les mêmes. Pour aller plus loin, il conviendrait de demander à l'utilisateur de coopérer en lui demandant de saisir les deux valeurs qu'il veut additionner, comme dans l'exemple suivant :

```
var firstNumber = prompt("Entrez le premier nombre");
var secondNumber = prompt("Entrez le deuxième nombre");
var total = Number(firstNumber) + Number(secondNumber);
alert("Le résultat est : "+total);
```

Procédons par étapes :

1. **Ouvrez votre éditeur de code et créez un modèle HTML de base.**
2. **Entre les balises <body> et </body>, insérez une balise ouvrante <script>, puis une balise fermante </script>.**
3. **Entrez l'exemple de code précédent entre les balises de script.**

   Votre document devrait ressembler à ceci

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
  </head>
  <body>
  <script>
    var firstNumber = prompt("Entrez le premier nombre");
    var secondNumber = prompt("Entrez le deuxième nombre");
    var total = Number(firstNumber) + Number(secondNumber);
    alert("Le résultat est : "+total)
  </script>
  </body>
</html>
```



La ligne qui débute par `<meta` est indispensable pour que les caractères accentués soient correctement affichés par le navigateur.

4. **Sauvegardez votre document HTML (nous l'avons appelé addtwo.html).**
5. **Ouvrez le document HTML dans votre navigateur Web.**

   Celui-ci va vous demander de saisir un premier nombre (voir la Figure 3.1).

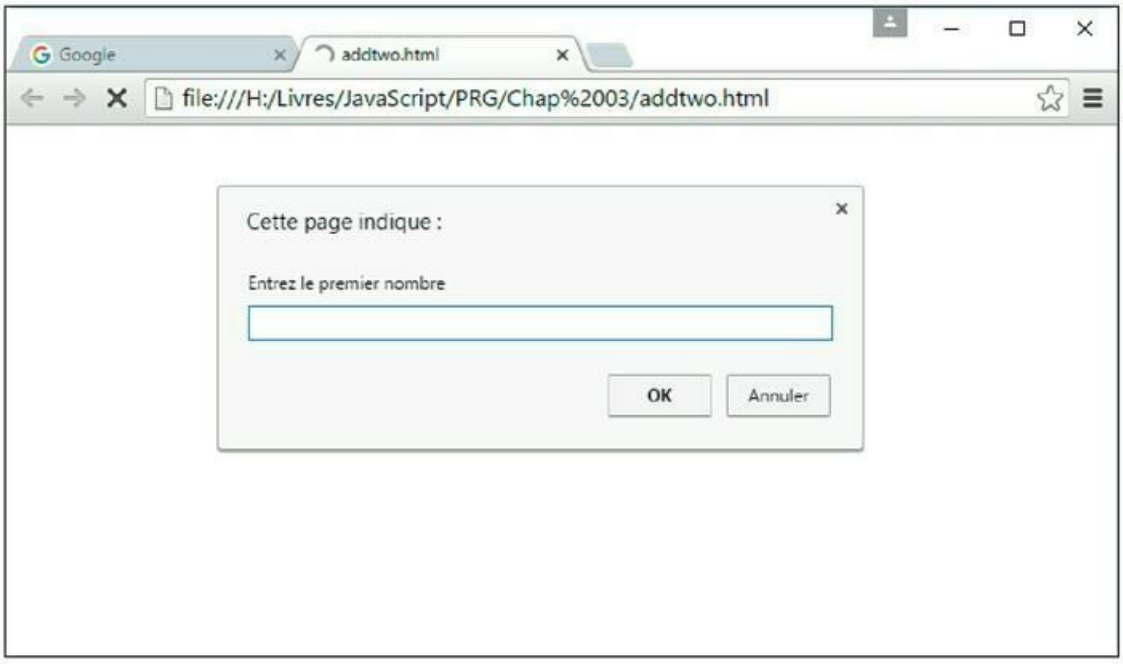


FIGURE 3.1 : Ce programme d'addition généraliste demande à l'utilisateur de saisir deux nombres.

6. **Entrez le premier nombre.**

   Le programme vous en demande maintenant un second.

7. **Entrez le second nombre.**

   Lorsque vous cliquez ensuite sur OK, le résultat de l'addition est affiché à l'écran.

Déclarer des variables

Déclarer une variable est un terme technique qui est utilisé pour décrire le processus consistant à créer une nouvelle variable dans un programme. Vous pourriez d'ailleurs aussi l'appeler *initialisation*. Créer une variable, déclarer une variable, ou initialiser une variable, tout cela veut dire la même chose.

Les variables peuvent être créées de deux manières dans JavaScript :

» **En utilisant le mot-clé `var` :**

```
var maVariable ;
```

Une variable déclarée de cette manière aura une valeur initiale non définie, à moins de spécifier explicitement celle-ci, comme dans :

```
var monNom = "Chris" ;
```

» **Sans le mot-clé `var` :**

```
monNom = "Chris" ;
```

Lorsque vous définissez ainsi une variable, elle devient une variable *globale* (nous allons définir ce mot dans la prochaine section).



Notez l'emploi des apostrophes autour du nom dans les exemples ci-dessus. Elles indiquent que la variable doit être traitée comme du texte, et non comme un nombre, un mot-clé JavaScript ou une autre variable. Voyez la section sur les types de données, plus loin dans ce chapitre, pour plus d'informations sur l'usage des apostrophes.

Comprendre la portée des variables

Comment et quand vous déclarez une variable détermine comment et où votre programme pourra l'utiliser. Ce concept est appelé la *portée de la variable*.

Dans JavaScript, il existe deux types de portée :

- Les *variables globales* peuvent être utilisées partout à l'intérieur d'un programme.
- Les *variables locales* sont créées à l'intérieur d'une section protégée qui est contenue dans le programme, ce que l'on appelle une *fonction*.

QUAND ÉGAL N'EST-IL PAS ÉGAL ?

Dans un langage raisonnablement humain, on pourrait être tenté d'écrire `var monNom égale Chris`, ou encore `var monNom est Chris`. Cependant, cette interprétation n'est pas correcte en programmation.

Reprenons notre exemple :

```
var monNom = "Chris" ;
```

Le caractère qui ressemble à un signe égal (d'accord, c'en est un) s'obtient en tapant sur la touche correspondante du clavier. Mais cela n'est en quelque sorte que symbolique, et donc à ne pas prendre au pied de la lettre. En programmation, ce signe d'égalité s'appelle en fait l'*opérateur d'affectation*.

Il est vital de comprendre cette distinction entre affectation et égalité :

- L'opérateur d'affectation définit la chose qui se trouve à sa gauche comme contenant la chose qui est placée à sa droite. L'exemple ci-dessus doit donc être lu comme ceci : « Créer la variable appelée monNom et lui attribuer comme valeur Chris » (bien sûr, vous pouvez l'énoncer un peu différemment si vous le souhaitez).
- Le mot `égal` s'emploie généralement pour comparer les deux valeurs qui l'entourent, et décider si elles sont ou non identiques. En JavaScript, cet *opérateur de comparaison* s'écrit ==.



En règle générale, l'utilisation de variables locales est préférable à l'emploi de variables globales, car leur portée plus limitée réduit les risques de remplacement accidentel du contenu d'une variable par celui d'une autre variable portant le même nom.

L'emploi de variables globales dans vos programmes peut rendre les problèmes difficiles à localiser et à réparer. Par conséquent, nous vous recommandons de toujours créer vos variables en les faisant précéder du mot-clé `var`. Si vous avez vraiment besoin d'une variable globale, il est toujours possible de la créer en faisant appel au mot-clé `var`, et c'est la méthode que nous vous recommandons d'employer.

L'HISTOIRE TRAGIQUE DU VAR MANQUANT

Il n'y a jamais de raison qui puisse justifier le fait de créer une variable sans utiliser le mot-clé `var`, et l'omettre provoque le plus souvent des problèmes difficiles à détecter. Si vous ne le spécifiez pas, vous allez simplement donner l'impression de l'avoir oublié. Adoptez tout de suite le bon réflexe !

L'exemple qui suit montre le type de problème et de confusion qui peut se produire lorsque le mot-clé `var` est omis. Il montre également l'emploi d'un outil de programmation plus avancé, appelé une *fonction*, que nous traiterons en détail dans le Chapitre 7. Pour faire court, à ce stade, les fonctions vous permettent d'insérer de petits programmes dans un programme plus important.

Dans ce premier exemple, le programmeur voudrait définir une variable appelée `movie`, qui soit globale, et une autre variable distincte portant le même nom, mais qui soit valide uniquement dans la fonction appelée `showBadMovie`. Il s'agit de quelque chose de parfaitement légitime et, dans des conditions normales, la variable `movie` déclarée dans la fonction n'affecterait pas la variable globale. Cependant, si le mot-clé `var` est omis ou oublié dans cette déclaration, les ennuis commencent.

```
var movie = "Le Parrain";

    function showGoodMovie () {
      alert (movie + " est un bon film !");
    }

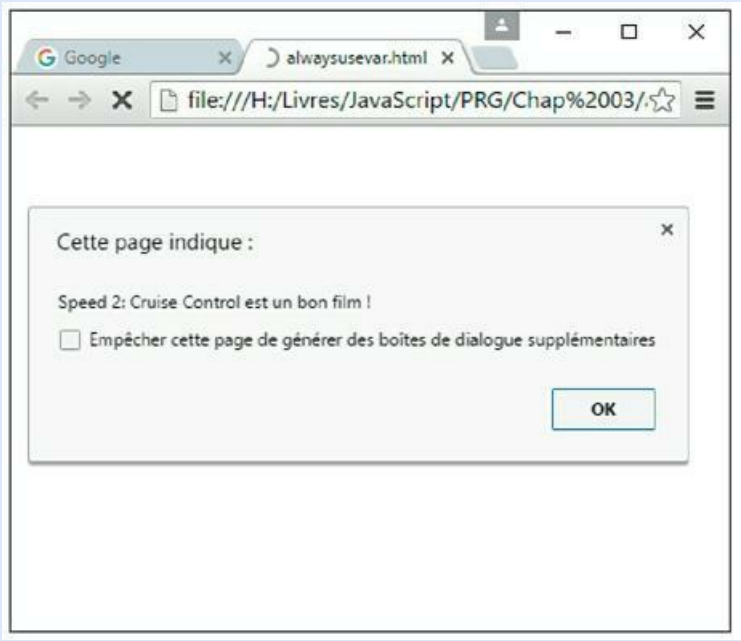
    function showBadMovie () {
      movie = "Speed 2: Cruise Control";
      alert (movie + " est un mauvais film");
  }
```

Remarquez que le mot-clé `var` est omis devant le nom de la variable `movie` dans la fonction `showBadMovie`. JavaScript suppose que vous voulez remplacer la variable globale `movie` par la valeur définie ici, plutôt que de créer une variable locale dans la fonction. Le résultat est potentiellement désastreux !

```
showGoodMovie () ; // affiche "Le Parrain est un bon film ! "

   showBadMovie (); // affiche "Speed 2: Cruise Control est un mauvais film
!"

   // Oh non ! La variable globale est maintenant Speed 2: Cruise Control,
 donc plus le nom du bon film !
showGoodMovie (); // affiche "Speed 2: Cruise Control est un bon film !"
```



Nommer les variables

Les noms des variables peuvent débuter par les caractères suivants :

- Une lettre minuscule ou majuscule
- un trait de soulignement (_)
- Un signe dollar ($)

Pour tous les projets de ce livre, je vous propose d'effectuer le montage sur une plaque d'essai sans soudure, car c'est plus facile. Si vous voulez ensuite rendre un projet permanent, ou si vous avez besoin de le protéger, il faudra prévoir un boîtier de dimensions suffisantes. Une fois le montage terminé, il vous suffit de prendre les dimensions et d'acquérir un boîtier approprié.

Même si les deux dernières options sont parfaitement licites, il est tout de même préférable de commencer par une lettre. L'emploi de caractères spéciaux rend souvent le code plus difficile à lire et à comprendre, surtout si vous débutez en JavaScript.

Après le premier caractère, vous pouvez utiliser n'importe quelle lettre ou n'importe quel chiffre dans le nom de votre variable. Par contre, les espaces sont interdites, de même que les opérateurs mathématiques ou les signes de ponctuation (le trait de soulignement mis à part).

QUELQUES CONSEILS POUR CRÉER DE BONS NOMS DE VARIABLES

Bien que JavaScript vous donne une très large liberté pour créer les noms de vos variables, il est préférable de définir vos propres règles avant même de commencer à programmer. Par exemple, allez-vous utiliser comme initiale une minuscule ou une majuscule ? Allez-vous aussi insérer des traits de soulignement dans les noms ? Plus votre code devient complexe, et plus l'importance d'un système de dénomination cohérent devient évident.

Heureusement, vous n'êtes pas seul lorsque vous devez prendre ce genre de décision. La plupart des professionnels ont un guide des bonnes pratiques que tout amateur qui se respecte devrait suivre. Voyons cela d'un peu plus près :

- Ne définissez pas des noms trop courts ! Utiliser une seule lettre, ou du moins un nom qui n'a pas de sens, est une très mauvaise habitude.
- Utilisez des noms comportant plusieurs mots de manière à être aussi précis que possible.
- Dans ce cas, placez les adjectifs qui servent à préciser le nom de votre variable sur la gauche de celui-ci, comme dans `vertSerpent`.

Choisissez une méthode qui convienne, et restez cohérent. Pour associer deux mots dans un nom de variable, vous avez le choix entre utiliser une lettre majuscule, comme dans `monNom`, ou un trait de soulignement, comme dans `mon_Nom` JavaScript est un langage très souple, et les deux méthodes sont parfaitement acceptables, même si la première est la plus répandue.

Certains mots ne peuvent pas être utilisés dans les noms des variables, pas plus que dans ceux des fonctions, des méthodes, des étiquettes de boucle ou des noms d'objets. En voici la liste :

abstract	instanceof
boolean	int
break	interface
byte	long
case	native
catch	new
char	null
class	package
const	private
continue	protected
debugger	public
default	return
delete	short
do	static
double	super
else	switch
enum	synchronized
export	this
extends	throw
false	throws
final	transient

finally	true
float	try
for	typeof
function	var
goto	void
if	volatile
implements	while
import	with
in	



N'oubliez jamais que JavaScript est sensible à la casse. En d'autres termes, une variable appelée `monNom` et une autre appelée `monnom` seront considérées par JavaScript comme étant différentes.

Les noms des variables sont en réalité des *identificateurs*. Vous devriez donc les nommer systématiquement d'une manière qui soit à la fois précise et significative. Les conventions que définissent les programmeurs peuvent parfois se traduire par des noms très longs. Mais, au final, un nom long qui définit clairement une variable est toujours préférable à un nom court dont la signification reste vague.



Bien entendu, il existe des limites quant à la longueur des noms, limites au-delà desquelles votre vie deviendrait plus compliquée. Si vingt caractères vous suffisent pour nommer une variable d'une manière qui la décrive parfaitement, c'est parfait. Mais si vous créez un nom dans le genre `nomDeLaPersonneQuiVientDeRemplirLeFormulaireSurMonSite`, vous devriez peut-être vous demander comment vous simplifiez l'existence (et celle des personnes qui auront besoin de travailler avec votre code) en le raccourcissant plus ou moins. Par exemple, `clientFormulaireRempli` est peut-être tout à fait suffisant (quoique `cFR` soit bien plus court, mais aussi nettement moins explicite).

Créer des constantes avec le mot-clé const

Dans certains cas, votre programme peut avoir besoin de variables dont le contenu ne doit pas être changé. On parle alors de *constantes*. Une constante est définie à l'aide du mot-clé `const`. Par exemple :

```
const heightOfTheEmpireStateBuilding = 1454;
const speedOfLight = 299792458;
const numberOfProblems = 99;
const meanNumberofBooksReadIn2014 = 12;
```

Les constantes suivent les mêmes règles que les autres variables. En revanche, leur contenu ne peut pas être modifié au cours de leur durée de vie (c'est-à-dire, celle du script où elles sont définies).

Travailler avec les types de données

Le *type de donnée* d'une variable est le genre d'information qu'elle contient, ce qui détermine également les opérations que cette variable est susceptible de se voir appliquer. Par exemple, le nombre 10, placé dans une phrase, est très différent du même nombre 10 utilisé dans une équation. Les types de données correspondent donc à

la manière dont JavaScript interprète ce qui doit être pris pour un mot, de ce qui peut être traité dans une expression mathématique.

Si vous réfléchissez à tous les types de données que vous rencontrez dans votre vie quotidienne, vous comprendrez vite qu'il s'agit d'un sujet très compliqué. Par exemple, quel est le rapport entre un graphique, une recette, une histoire, un article de presse, et ainsi de suite. La plupart relèvent du langage courant, mais comment leur donner un sens, une cohérence ? Les créateurs de JavaScript ont été généreux, et ils ont décidé de vous simplifier l'existence. En fait, JavaScript se contente de cinq types de données de base.

Au-delà, JavaScript est aussi appelé un *langage pauvrement typé*. Autrement dit, vous n'avez même pas besoin de savoir à l'avance si la variable que vous créez va contenir un mot, un paragraphe, un nombre, ou encore un autre type de donnée.

Cela ne signifie en rien que JavaScript n'est pas capable d'effectuer la distinction entre les mots et les nombres. JavaScript se débrouille très bien avec les uns et avec les autres, et il est capable de se représenter le type de donnée que vous enregistrez dans vos variables bien en amont de la scène.



JavaScript reconnaît cinq types de données, dits primitifs.

De son côté, le langage de programmation C++ accepte et reconnaît pas moins de douze types de données différents !

Les données numériques

En JavaScript, les nombres sont enregistrés sous la forme de valeurs en virgule flottante sur 64 bits. Cela signifie que les nombres peuvent valoir de $5e^{-234}$ (soit 5 suivi de 324 zéros) jusqu'à $1.7976931348623157e^{+308}$ (déplacez le point décimal de 308 positions vers la droite pour voir ce nombre gigantesque). Tout nombre peut posséder ou non un point décimal (les langages de programmation ne reconnaissent pas nos virgules). Contrairement à d'autres langages de programmation, JavaScript ne fait pas la différence entre les nombres entiers et les nombres décimaux.



Le plus grand nombre que JavaScript est capable d'utiliser est représenté ci-après sans notation scientifique :

179769313486231570000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000

Lorsque vous déclarez une variable numérique, vous utilisez tous les éléments suivants :

- le mot-clé `var` ;
- le nom que vous donnez à votre variable ;
- l'opérateur d'affectation ;
- un nombre (ou même une équation qui se résout en un nombre) ;
- un point-virgule.

Voici quelques exemples de déclarations de variables :

```
var nombreDePattes = 4;
var populationEspagne = 47200000;
var combienDeParts = 8;
```

Fonction Number

JavaScript dispose d'une fonction appelée `Number` pour convertir des valeurs en nombres. Pour utiliser cette fonction, vous placez simplement la valeur que vous voulez convertir (ou une variable qui contient cette valeur) entre parenthèses à la suite du nom de la fonction.

La fonction `Number` produit quatre types de sorties :

- Les nombres qui sont formatés sous la forme d'une chaîne de caractères sont convertis en valeurs numériques qui peuvent être utilisées dans des calculs, comme ceci :

  ```
  Number ("42") // retourne le nombre 42
  ```

- Les chaînes de caractères qui ne peuvent pas être converties en nombres retournent la valeur `NaN`, comme ceci :

  ```
  Number ("oeufs") // retourne NaN
  ```

- La valeur booléenne `true` (vrai) renvoie le nombre 1, comme ceci : `Number (true) // retourne 1`

- La valeur booléenne `false` (faux) renvoie le nombre 0, comme ceci :

  ```
  Number (false) // retourne 0
  ```

Fonction parseInt ()

Du point de vue de JavaScript, tous les nombres sont des valeurs en virgule flottante. Cependant, vous pouvez faire appel à la fonction `parseInt ()` pour demander à JavaScript de ne prendre en compte que la partie entière de la valeur numérique (ce qui se trouve avant le point décimal), et donc d'éliminer le reste du nombre. Par exemple :

```
parseInt (100.33) ; // retourne 100
```

Fonction parseFloat ()

La fonction `parseFloat ()` demande à JavaScript de traiter un nombre comme étant décimal. Vous pouvez même vous en servir pour convertir une chaîne en nombre. Par exemple :

```
parseFloat("10"); // retourne 10
parseFloat(100.00); //retourne 100.00
parseFloat("10"); //retourne 10
```

Exemples

Vous pouvez maintenant jouer un peu avec quelques nombres et les fonctions numériques que nous venons de décrire. Essayez d'entrer les expressions suivantes dans la console JavaScript de votre navigateur Chrome pour voir les résultats qu'elles produisent.



Vous pouvez ouvrir la console JavaScript de Chrome en appuyant sur les combinaisons de touches Command + Option + J (Mac) ou Ctrl + Maj + J (Windows).

```
1 + 1
3 * 3
```

```
parseFloat("839");
parseInt("33.333333");
12 + "12"
"12" + 12
"12" * 2
```



Les variables numériques doivent être déclarées sans guillemets. Ainsi, “10” n’est pas la même chose que 10. Dans le premier cas, il s’agit d’une chaîne de caractères (voyez à ce sujet la prochaine section). Si vous déclarez par erreur une variable numérique en utilisant des guillemets, vous obtiendrez des résultats imprévisibles.

Si vous avez suivi les exemples ci-dessus, vous avez pu remarquer certains comportements étranges. Ainsi, lorsque vous ajoutez “12” (une chaîne) et 12 (un nombre), le résultat est “1212” (une chaîne). Mais si vous multipliez “12” (une chaîne) par 2 (un nombre), la réponse est 24 (un nombre). C’est qu’il est malin, JavaScript !

Dans le premier cas, JavaScript suppose que, du fait que le premier élément est une chaîne de caractères, vous voulez en fait mettre les deux valeurs bout à bout. Il convertit donc le nombre en chaîne et traite donc le symbole plus comme étant un *opérateur de concaténation*.

Dans le second exemple, JavaScript sait parfaitement qu’il n’est pas possible de multiplier des chaînes de caractères. Dans ce cas, il commence par convertir la chaîne en nombre, puis il effectue la multiplication. Mais que va-t-il se passer si vous tentez de multiplier deux chaînes ensemble ?

```
"sassafras" * "orange"
```

Le résultat est `NaN` (qui signifie *Not a Number*, pas un nombre). Comme il n’existe aucun moyen de convertir ces deux noms en valeurs numériques, JavaScript lève les bras au ciel et déclare forfait.

Les données de type chaîne

Les chaînes peuvent être constituées de n’importe quels caractères :

- lettres ;
- chiffres ;
- signes de ponctuation (comme des virgules ou des points) ;
- caractères spéciaux qui seront alors précédés d’une barre oblique inverse (\).

Certains caractères, comme les guillemets ou les apostrophes, ont une signification particulière en JavaScript, ou nécessitent une combinaison spécifique de caractères pour pouvoir être représentés dans une chaîne (comme une tabulation ou un saut de ligne). Le Tableau 3.1 liste les *caractères spéciaux* que vous pouvez utiliser en JavaScript.

Tableau 3.1 : Les caractères spéciaux de JavaScript.

Code	Ce qu’il représente
\’	apostrophe
\”	guillemet
\\	barre oblique inverse
\n	nouvelle ligne
\r	retour chariot

\t	tabulation
\b	retour arrière
\f	saut de page

Vous créez une variable chaîne en l'entourant d'apostrophes ou de guillemets, comme ceci :

```
Var maChaine = "Bonjour, je suis une chaîne." ;
```

L'emploi de l'apostrophe ou du guillemet est indifférent, dès lors que vous utilisez le même caractère au début et à la fin de la chaîne.

Si vous vous servez d'apostrophes, vous pouvez inclure sans aucun problème des guillemets à l'intérieur de la chaîne. De même, si vous délimitez la chaîne par des guillemets, l'emploi d'apostrophes à l'intérieur de celle-ci est parfaitement licite.



En revanche, il n'est pas possible de réemployer le marqueur de début de chaîne à l'intérieur de celle-ci. Sinon, JavaScript pensera que votre chaîne se termine à cet emplacement, et il ne comprendra rien à ce qui suit.

Caractères d'échappement

Comme il n'est pas possible d'inclure des apostrophes (ou des guillemets) à l'intérieur d'une chaîne également délimitée par des apostrophes (ou des guillemets), il faut tricher un petit peu. La solution consiste à faire précéder le caractère d'une barre oblique inverse (\). C'est ce que l'on appelle un *échappement*, ou encore une *séquence d'échappement*.

Fonctions de chaîne

JavaScript contient de nombreuses fonctions utiles pour travailler avec les chaînes et pour les convertir.

Voici une liste des fonctions de chaîne les plus couramment utilisées dans les programmes :

» `charAt ()` : Renvoie le caractère qui se trouve à la position spécifiée. Notez que le dénombrement des caractères débute à la position 0. Exemple :

```
var uneChaine = 'JavaScript est génial ! ' ;
console.log (unChaine.charAt (3)) ;
// retourne a
```

» `concat ()` : Combine une ou plusieurs chaînes en une nouvelle chaîne.

Exemple :

```
var uneChaine = 'JavaScript est génial ! ' ;
console.log (uneChaine.concat (' Nous aimons JavaScript ! ')) ;
// retourne JavaScript est génial ! Nous aimons JavaScript !
```

» `indexOf ()` : Recherche et renvoie la position de la première occurrence du caractère recherché, ou d'une sous-chaîne à l'intérieur de la chaîne.

Exemple :

```
var uneChaine = 'JavaScript est génial ! ' ;
```

```
console.log (uneChaine.indexOf (génial') ;
// retourne 14
```

» `split ()` : Partage une chaîne en un tableau de sous-chaînes. Exemple :

```
var uneChaine = ‘JavaScript est génial ! ' ;
console.log (uneChaine.split (‘g')) ;
// retourne ["JavaScript est ", "énial ! "]
```

» `substr ()` : Extrait une partie de la chaîne commençant à la position indiquée et possédant la longueur spécifiée. Exemple :

```
var uneChaine = ‘JavaScript est genial ! ' ;
console.log (uneChaine.substr (2,5)) ;
// retourne vaScr
```

» `substring ()` : Extrait les caractères de la chaîne compris entre deux positions spécifiées. Exemple :

```
var uneChaine = ‘JavaScript est genial ! ' ;
console.log (uneChaine.substring (2,5)) ;
// retourne vaS
```

» `toLowerCase ()` : Convertit tous les caractères de la chaîne en minuscules.

Exemple :

```
var uneChaine = ‘JavaScript est génial ! ' ;
console.log (uneChaine.toLowerCase ()) ;
// retourne javascript est génial !
```

» `toUpperCase ()` : Convertit tous les caractères de la chaîne en majuscules.

Exemple :

```
var uneChaine = ‘JavaScript est génial ! ' ;
console.log (uneChaine.toUpperCase ()) ;
// retourne JAVASCRIPT EST GÉNIAL !
```

Type de donnée booléen

Les variables booléennes ne peuvent contenir que l’une des deux valeurs `true` (vrai) ou `false` (faux).



Le terme *booléen* provient du nom du mathématicien anglais George Boole (1815-1864) qui a créé un système algébrique de logique.

Les variables booléennes sont souvent employées pour enregistrer le résultat de comparaisons. Vous pouvez trouver la valeur booléenne d'une comparaison, ou convertir une valeur en un booléen, en utilisant la fonction JavaScript judicieusement appelée `Boolean ()`. Par exemple :

```
var estPlusGrand = Boolean (3 > 20);
alert (estPlusGrand); // retourne false
var estIdentique = Boolean ("tigre" === "Tigre");
alert (estIdentique); // retourne false
```

Le résultat produit par la conversion en JavaScript d'une certaine valeur en une valeur booléenne dépend de son contenu :

- Les valeurs suivantes sont toujours évaluées comme false par la fonction `Boolean ()` :
  - `NaN`
  - indéfini
  - `0` (valeur numérique 0)
  - `-0`
  - `""` ou '' (chaîne vide)
  - `false`
- Tout ce qui ne relève pas des cas précédents est évalué avec la valeur booléenne `true`. Par exemple :
  - 74
  - "Eva"
  - "10"
  - "NaN"



La chaîne `"0"` est différente de la valeur numérique `0`. Cette dernière donne une valeur booléenne `false`, tandis que la première retourne une valeur booléenne `true`.

Les valeurs booléennes sont essentiellement utilisées avec des expressions conditionnelles. Le programme qui suit crée une variable booléenne, puis il la teste dans une instruction `if / then` (nous expliquerons de quoi il s'agit dans le Chapitre 5) :

```
var b = true;
if (b == true) {
  alert ("C'est vrai !");
  } else {
    alert ("C'est faux.");
}
```



Les valeurs booléennes sont écrites sans apostrophes ou guillemets, comme ceci :

```
var maVar = true ;
```

D'un autre côté, une instruction comme var `maVar = "true"` produirait simplement une variable chaîne.

Le type de donnée NaN

NaN est l'abréviation de *Not a Number* (pas un nombre). C'est le résultat que vous obtenez si vous essayez de faire des calculs avec une chaîne, ou bien si un calcul échoue ou ne peut pas être effectué. Par exemple, il est impossible de calculer la racine carrée d'un nombre négatif. Tenter de le faire produira comme résultat NaN.



Un cas classique d'instruction produisant comme résultat NaN est une tentative d'opération mathématique sur une chaîne qui ne peut pas être convertie en valeur numérique.

Le type de donnée indéfini

Même si vous créez une variable JavaScript sans rien lui affecter, elle possède tout de même une valeur par défaut. Cette valeur est "`undefined`" (indéfini).

Chapitre 4

Comprendre les tableaux

DANS CE CHAPITRE :

- **Identifier et définir des tableaux**
- **Construire des tableaux**
- **Passer de la 2D à des tableaux multidimensionnels**
- **Travailler avec les éléments des tableaux**
- **Utiliser les fonctions et les propriétés des tableaux**

« Je suis grand. Je contiens des multitudes. »

Walt Whitman

Les tableaux sont une partie fondamentale de tout langage de programmation. Dans ce chapitre, vous allez découvrir ce qu'ils sont, comment les utiliser, et ce qui les rend différents dans JavaScript par rapport à d'autres langages de programmation. Vous travaillez avec des tableaux pour créer des listes, les ordonner, ainsi que pour ajouter ou supprimer des éléments dans des listes.



N'oubliez pas de visiter le site Web compagnon de ce livre pour y télécharger les exemples de ce chapitre !

Faire une liste

Dans les chapitres précédents, nous avons appris à travailler avec des variables ne contenant qu'une seule donnée, comme `var monNom = "Chris"`, `var unNombre = "3"`, et ainsi de suite. Mais il existe des tas de situations en programmation (comme dans la vraie vie) où vous voudriez pouvoir enregistrer des données similaires sous un même nom. Considérons par exemple les types de listes suivants :

- Une liste de vos artistes préférés
- Un programme qui sélectionne et affiche une citation choisie dans une liste chaque fois qu'il est exécuté
- Une liste d'adresses pour envoyer des cartes de vœux
- Une liste de vos albums de musique préférés de l'année
- Une liste des dates d'anniversaire des membres de votre famille et de vos amis
- Une liste de courses
- Une liste de tâches à accomplir
- Une liste des bonnes résolutions que vous allez prendre pour la nouvelle année

Avec des variables comme celles que nous avons rencontrées dans le Chapitre 3, vous devriez créer et gérer de multiples dénominations pour arriver à stocker les informations voulues. Par exemple :

```
var artist1 = "Alphonse Mucha";
var artist2 = "Chiara Bautista";
var artist3 = "Claude Monet";
var artist4 = "Sandro Botticelli";
var artist5 = "Andy Warhol";
var artist6 = "Wassily Kadinski";
var artist7 = "Vincent Van Gough";
var artist8 = "Paul Klee";
var artist9 = "William Blake";
var artist10 = "Egon Schiele";
var artist11 = "Salvador Dali";
var artist12 = "Paul Cezanne";
var artist13 = "Diego Rivera";
var artist14 = "Pablo Picasso";
```

Cette approche pourrait peut-être suffire sur le court terme, mais elle va très rapidement vous poser bien des problèmes. Supposons ainsi que vous vouliez trier cette liste dans l'ordre alphabétique, et placer chaque artiste dans le nom de variable correspondant à sa position. Vous allez donc devoir retirer Mucha dans la première variable (disons, pour le placer dans une variable temporaire), puis déplacer Bautista vers cette variable. La variable `artist2` sera alors disponible pour y enregistrer Blake, mais n'oubliez pas que Mucha se trouve toujours dans un conteneur temporaire ! Comme Blake est passé de la position 9 à la position 2, son emplacement d'origine est maintenant disponible. Et ainsi de suite. Bref, il est facile de comprendre que vous auriez plus vite fait en écrivant tous les noms sur une feuille de papier… Créer une liste de cette manière devient très vite compliqué, confus, et pour tout dire ingérable.

Heureusement, JavaScript (comme tous les autres langages de programmation qui se respectent) prend en charge la création de variables contenant des valeurs multiples, ce que l'on appelle des *tableaux*.

Les tableaux constituent un moyen permettant d'enregistrer des groupes de données censées présenter une certaine cohérence à l'intérieur d'une unique variable. Avec eux, vous pouvez créer des listes contenant aussi bien des chaînes que des nombres, des valeurs booléennes, de même que tout autre type de donnée, et même d'autres tableaux !

Notions de base sur les tableaux

Un tableau contient des *éléments de tableau*. Un élément de tableau est formé à partir du nom du tableau lui-même, suivi d'un numéro d'indice qui est placé entre des crochets droits. Ce sont ces valeurs individuelles qui constituent les éléments du tableau. Les tableaux utilisent des nombres (les *indices*) pour accéder à ces éléments. L'exemple qui suit illustre ces notions :

```
monTableau[0] = "ballon jaune";
monTableau[1] = "ballon rouge";
monTableau[2] = "ballon bleu";
monTableau[3] = "ballon rose";
```

Ici, l'élément du tableau d'indice 0 contient la valeur `"ballon jaune"`. L'élément d'indice 3 a pour valeur `"ballon rose"`. Comme pour toute autre variable, vous pouvez donner à vos tableaux des noms conformes aux

règles de JavaScript. Grâce à ce système d'indices, JavaScript vous donne la possibilité de regrouper une liste pratiquement illimitée de valeurs sous un même nom.



L'expression « pratiquement illimitée » a tout de même ses limites, même si vous avez peu de chances de les atteindre un jour. Cette limite est de 4.294.967.295 éléments.

En plus des règles et méthodes de dénomination des variables (si nécessaire, revoyez à ce sujet le Chapitre 3), les tableaux ont quelques autres propriétés spéciales avec lesquelles vous devez vous familiariser :

- L'indice du premier élément d'un tableau est 0.
- Les tableaux peuvent stocker des données de n'importe quel type.

L'indice de base d'un tableau, c'est 0

JavaScript n'a pas de doigts. Par conséquent, il n'a pas besoin de dénombrer les choses en partant de 1. Le premier élément de tout tableau JavaScript reçoit comme indice 0 (voir la Figure 4.1).

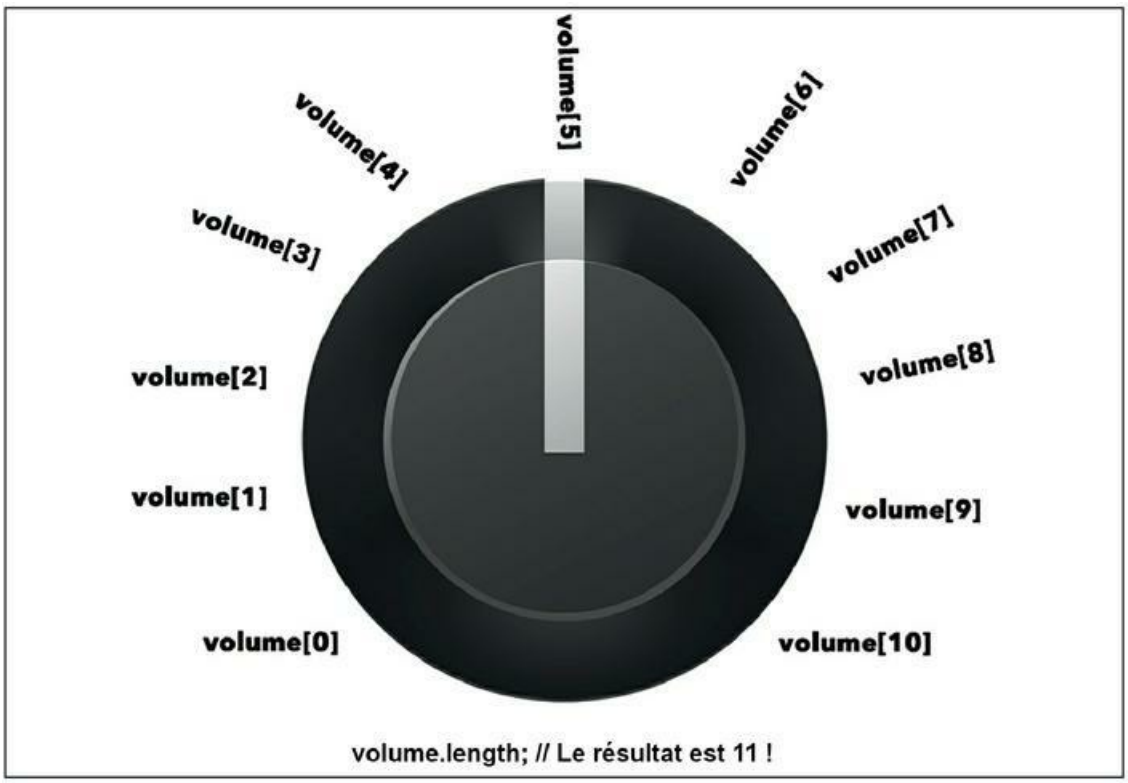


FIGURE 4.1 : JavaScript est un peu comme un bouton de volume. Il commence à compter à partir de zéro !

Dit autrement, l'élément `monTableau[3]` est en fait le quatrième élément du tableau.

Cette technique de numérotation à partir de zéro est souvent une source de confusion et d'erreur pour les programmeurs débutants. Mais tout cela finit vite par devenir naturel. En fait, ce système permet de simplifier le travail du programme et vous habitue au fait que, si vous tournez le bouton du volume sur le minimum, le niveau du son est réduit à zéro !

Les tableaux peuvent contenir n'importe quel type de donnée

Chaque élément d'un tableau peut stocker un type de donnée quelconque (les types de données JavaScript sont décrits dans le Chapitre 3), mais aussi un autre tableau. Plus largement encore, ces éléments acceptent des fonctions

et des objets JavaScript (voyez à ce sujet les Chapitres 7 et 8).

En fait, plusieurs éléments d'un même tableau sont parfaitement capables de contenir des données de type très différent, comme l'illustre l'exemple du Listing 4.1.

LISTING 4.1 : Différents types de données dans un même tableau.

```
item[0] = "pomme";
item[1] = 4+8;
item[2] = 3;
item[3] = item[2] * item[1];
```

Créer des tableaux

JavaScript fournit deux méthodes pour la création de nouveaux tableaux :

- le mot-clé `new` ;
- la notation littérale.

Utiliser le mot-clé new

Cette méthode utilise le mot-clé `new` placé devant `Array` (tableau) pour créer un nouveau tableau et y ajouter des valeurs, comme ceci :

```
var nomsChats = new Array ("Minou", "Chatounet", "Mr. Chat") ;
```

Cette méthode de création est parfaitement valable, et vous la rencontrerez forcément au cours de votre carrière de programmeur.



Des nombreux experts en JavaScript déconseillent pourtant l'utilisation du mot-clé `new`. Le principal problème avec cette méthode, c'est tout simplement d'oublier de spécifier ce mot-clé, ce qui peut changer dramatiquement la façon dont votre programme se comporte.

La méthode littérale

Une manière plus simple et plus sûre pour créer des tableaux consiste à employer ce que l'on appelle la méthode littérale de notation. Voici comment les choses se présentent :

```
var nomsChiens = ["Médor", "Milou", "Dr. Spock"] ;
```

Et c'est tout ! L'utilisation des crochets droits sans autre précision indique à JavaScript qu'il s'agit d'un tableau. De cette manière, vous avez moins de risques d'oublier quelque chose. Cette notation est également plus courte, ce qui n'est pas non plus à négliger lorsque l'on doit saisir des centaines, voire des milliers, de lignes de code !

Remplir les tableaux

Vous pouvez ajouter des valeurs à un tableau au moment de sa création, ou simplement effectuer cette création, puis ajouter ultérieurement les éléments en fonction de vos besoins. En fait, le processus est le même que pour la création d'une quelconque variable, si ce n'est que vous devez spécifier l'indice de l'élément que vous voulez créer ou modifier. Le Listing 4.2 vous montre comment définir un tableau vide, puis comment le remplir.

LISTING 4.2 : Remplir un tableau vide.

```
var myJazz = [];
myJazz[0] = "Duke Ellington";
myJazz[1] = "Charlie Parker";
myJazz[2] = "Miles Davis";
```

Il n'est pas obligatoire d'ajouter les éléments dans l'ordre des indices. En d'autres termes, il est parfaitement légal en JavaScript de créer un élément ayant une position quelconque dans le tableau. En repartant de l'exemple précédent, nous pourrions tout à fait poursuivre la série en écrivant :

```
myJazz[99] = "John Coltrane" ;
```

En procédant ainsi, JavaScript va considérer que votre tableau contient 100 éléments, tous ceux qui sont placés entre `myJazz[2]` et `myJazz[99]` étant vierges (non définis).

Si vous ne nous croyez pas sur parole, il ne vous reste plus qu'à tester la dimension du tableau de la manière suivante :

```
myJazz.length // retourne 100
```

Même si vous n'avez en fait défini que quatre éléments, JavaScript vous indique que la longueur du tableau est égale à 100, puisque cette longueur est basée sur l'indice de rang le plus élevé, et non le contenu effectif du tableau.

Comprendre les tableaux multidimensionnels

Vous pouvez placer des tableaux dans d'autres tableaux. Mais l'histoire ne s'arrête pas là. JavaScript vous permet aussi de créer des tableaux de tableaux, ou encore des tableaux de tableaux de tableaux, et ainsi de suite. C'est ce que l'on appelle des *tableaux multidimensionnels*. Si vous avez des problèmes avec la 3D, vous frisez sans doute déjà le vertige sidéral.

En fait, un tableau multidimensionnel se définit plutôt simplement en ajoutant des crochets droits supplémentaires à la suite du nom de la variable. Par exemple :

```
var listeDeListes[0][0] ;
```

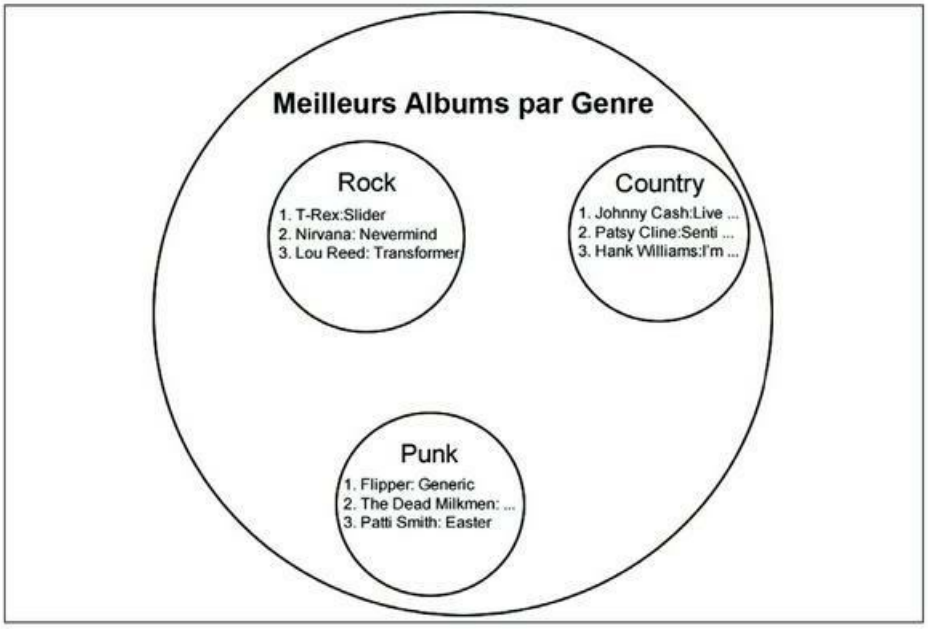


FIGURE 4.2 : Graphique représentant un tableau multidimensionnel.

Les tableaux multidimensionnels peuvent être difficiles à visualiser, notamment lorsque vous commencez à travailler avec eux. La Figure 4.2 illustre un exemple d'un tel tableau.

Vous pouvez également vous représenter un tableau multidimensionnel sous la forme d'un plan, ou encore de listes hiérarchisées. Par exemple :

Meilleurs Albums par Genre

1. Country
    1.1 Johnny Cash : Live at Folsom Prison
    1.2 Patsy Cline : Sentimentally Yours
    1.3 Hank Williams : I'm Blue Inside
2. Rock
    2.1 T Rex : Slider
    2.2 Nirvana : Nevermind
    2.3 Lou Reed : Transformer
3. Punk
    3.1 Flipper : Generic
    3.2 The Dead Milkmen : Big Lizard in my Backyard
    3.3 Patti Smith : Easter

Voici le code qui permettrait de créer un tableau multidimensionnel en partant de la Figure 4.2 :

```
var meilleursAlbumsParGenre = []
meilleursAlbumsParGenre[0] = "Country";
meilleursAlbumsParGenre[0][0] = "Johnny Cash:Live at Folsom Prison"
```

```
meilleursAlbumsParGenre[0][1] = "Patsy Cline:Sentimentally Yours";
meilleursAlbumsParGenre[0][2] = "Hank Williams:I'm Blue Inside";
meilleursAlbumsParGenre[1] = "Rock";
meilleursAlbumsParGenre[1][0] = "T-Rex:Slider";
meilleursAlbumsParGenre[1][1] = "Nirvana:Nevermind";
meilleursAlbumsParGenre[1][2] = "Lou Reed:Tranformer";
meilleursAlbumsParGenre[2] = "Punk";
meilleursAlbumsParGenre[2][0] = "Flipper:Generic";
meilleursAlbumsParGenre[2][1] = "The Dead Milkmen:Big Lizard in my
Backyard";
meilleursAlbumsParGenre[2][2] = "Patti Smith:Easter";
```

Accéder aux éléments d'un tableau

Vous pouvez accéder aux éléments d'un tableau exactement de la même manière que vous définissez celui-ci, autrement dit en spécifiant un indice placé entre des crochets droits. Par exemple, l'instruction suivante accède à l'élément de rang 2 d'un tableau appelé `monTableau` :

```
monTableau[2] ;
```

Dans le cas d'un tableau multidimensionnel, vous devez juste ajouter autant de crochets droits et d'indices qu'il est nécessaire pour atteindre l'élément voulu. Par exemple :

```
meilleursAlbumsParGenre[0][1]; // retourne "Patsy Cline:Sentimentally
Yours"
```

Pour tester ces techniques, suivez ces étapes :

1. **Lancez le navigateur Chrome et ouvrez la console JavaScript.**



Pour ouvrir la console de Chrome, vous pouvez utiliser les raccourcis Command + Option + J (Mac) ou Ctrl + Maj + J (Windows).

2. **Dans la console, tapez l'instruction suivante, puis appuyez sur la touche Entrée. Cela va créer un tableau appelé `longueurChaine` :**

```
var longueurChaine = [2,4,1.5,80] ;
```

3. **Tapez le nom du tableau suivi du numéro d'indice entre crochets afin de retrouver l'élément correspondant :**

```
longueurChaine[0];
longueurChaine[3];
longueurChaine[2];
```

---

4. **Entrez un numéro d'indice qui n'existe pas dans le tableau :**

```
longueurChaine[4] ;
```

Vous remarquez que cet élément est indiqué comme étant non défini (`undefined`).

5. **Tapez la commande suivante afin de créer une nouvelle variable qui contiendra la longueur totale des éléments de votre tableau :**

```
var longueurTotale = longueurChaine[0] +
      longueurChaine[1] + longueurChaine[2] + longueurChaine[3];
```

---

6. **Vous obtenez alors le résultat voulu en saisissant l'instruction suivante :**

```
longueurTotale ;
```

Parcourir un tableau

Comme vous l'imaginez, travailler avec de multiples valeurs en notant le nom du tableau et le numéro d'indice de chaque élément peut devenir un calvaire pour vos doigts. Heureusement, il existe des moyens plus simples pour travailler avec tous les éléments d'un tableau. La méthode la plus courante consiste à utiliser une construction de programmation appelée une *boucle*. Les boucles seront traitées dans le Chapitre 6.

Il est aussi possible de travailler avec les éléments d'un tableau en faisant appel aux fonctions dédiées de JavaScript.

Propriétés des tableaux

Vous pouvez obtenir certaines informations sur un tableau en accédant à ses propriétés. En JavaScript, vous utilisez pour cela ce que l'on appelle la *notation point*. Pour cela, vous tapez le nom du tableau, suivi d'un point, puis de la propriété à laquelle vous voulez accéder (nous reviendrons plus en détail sur les propriétés dans le Chapitre 8). Le Tableau 4.1 liste toutes les propriétés des tableaux JavaScript.

Tableau 4.1 : Les propriétés des tableaux en JavaScript.

Propriété	Description
prototype.	Permet l'ajout de propriétés et de méthodes à un objet Array
constructor	Une référence à la fonction ayant créé le prototype de l'objet Array.
length	Retourne ou définit le nombre d'éléments dans un tableau.

La propriété la plus couramment utilisée est `length`. Vous l'avez déjà vue en action. Elle permet d'obtenir le nombre d'éléments dans un tableau, que ceux-ci soient ou non définis. Par exemple :

```
var monTableau = [];
monTableau[2000];
monTableau.length; // retourne 2001
```

Vous pouvez également définir la valeur de cette propriété afin, par exemple, de tronquer un tableau :

```
monTableau.length; // retourne 2001
monTableau.length = 10;
monTableau.length; // retourne 10
```

Méthodes pour les tableaux

Les *méthodes* JavaScript dédiées aux tableaux (ou fonctions de tableau) fournissent des outils commodes pour travailler avec les tableaux et les manipuler. Le Tableau 4.2 liste toutes les méthodes associées aux tableaux en décrivant ce qu'elles font et les valeurs qu'elles produisent.

Tableau 4.2 : Méthodes dédiées aux tableaux dans JavaScript.

Méthode	Valeur de retour
`concat ()`	Un nouveau tableau formé du tableau courant concaténé avec un ou plusieurs autres tableaux, et/ou valeurs.
`every ()`	Renvoie `true` si chaque élément du tableau satisfait aux conditions de la fonction de test spécifiée.
`filter ()`	Un nouveau tableau contenant tous les éléments du tableau courant testés comme valant `true` par la fonction spécifiée.
`forEach ()`	Exécute la fonction spécifiée pour chaque élément du tableau.
`indexOf ()`	La première occurrence de la valeur spécifiée à l'intérieur du tableau. Renvoie -1 si la valeur n'est pas trouvée.
`join ()`	Joint tous les éléments du tableau en une seule chaîne.
`lastIndexOf ()`	La dernière occurrence de la valeur spécifiée à l'intérieur du tableau. Renvoie -1 si la valeur n'est pas trouvée.
`map ()`	Un nouveau tableau contenant le résultat produit sur chaque élément du tableau courant par la fonction spécifiée.
`pop ()`	Supprime le dernier élément d'un tableau.
`push ()`	Ajoute de nouveaux éléments à la fin d'un tableau.
`reduce ()`	Réduit deux valeurs d'un tableau en une seule valeur en leur appliquant la fonction spécifiée (de gauche à droite).
`reduceRight ()`	Réduit deux valeurs d'un tableau en une seule valeur en leur appliquant la fonction spécifiée (de droite à gauche).
`reverse ()`	Inverse l'ordre des éléments d'un tableau.
`shift ()`	Supprime le premier élément d'un tableau et retourne cet élément, la longueur du tableau étant modifiée en conséquence.
`slice ()`	Sélectionne une partie d'un tableau et retourne le résultat dans un nouveau tableau.
`some ()`	Retourne `true` si un ou plusieurs éléments du tableau courant satisfont à la fonction de test spécifiée.
`sort ()`	

	Retourne un tableau une fois les éléments du tableau courant triés (le tri par défaut est alphabétique et ascendant).
`splice ()`	Retourne un nouveau tableau formé des éléments qui ont été ajoutés ou supprimés d'un tableau donné.
`toString ()`	Convertit un tableau en chaîne de caractères.
`unShift ()`	Retourne un nouveau tableau avec une nouvelle longueur par ajout d'un ou plusieurs éléments.

Utiliser les méthodes de tableau

La syntaxe de ces méthodes varie de l'une à l'autre. Cependant, vous y accédez de la même manière que dans le cas des propriétés, c'est-à-dire en utilisant la notation point.



Pour consulter un guide de référence complet sur les méthodes de tableau JavaScript, avec des exemples utiles, visitez le site https://docs.webplatform.org/wiki/javascript/Array#Methods.

Le Listing 4.3 illustre quelques exemples d'utilisation de méthodes JavaScript courantes.

LISTING 4.3 : Quelques méthodes JavaScript courantes pour les tableaux.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
  <title>Méthodes courantes pour les tableaux</title>
</head>
<body>
  <script>
    var animals = ["tigre" , "ours"];
    var fruit = ["melon" , "orange"];
    var dishes = ["plat" , "bol" , "tasse"];

    var fruitsAndAnimals = fruit.concat(animals);
    document.write (fruitsAndAnimals + "<br>");

    var whereIsTheTiger = animals.indexof("tigre");
    document.write ("Le tigre a un numéro d'index de : " +
whereIsTheTiger +
"<br>");
  </script>
</body>
</html>
```

La Figure 4.3 illustre le résultat affiché dans la fenêtre d'un navigateur.



FIGURE 4.3 : Des méthodes de tableau JavaScript courantes en action.

Chapitre 5

Travailler avec les opérateurs, les expressions et les instructions

DANS CE CHAPITRE :

- **Lire et coder des expressions JavaScript**
- **Changer des valeurs avec les opérateurs d'affectation**
- **Penser logiquement avec les opérateurs de comparaison**
- **Faire des calculs mathématiques avec les opérateurs arithmétiques**
- **Agir au niveau du bit**
- **Les chaînes et leurs opérateurs**

« Bonjour, opérateur. Pourriez-vous me donner le nombre 9 ? »

The White Stripes

Les opérateurs, les expressions et les instructions sont, dans JavaScript comme dans tout autre langage de programmation, les briques à partir desquelles vous allez construire vos programmes. Ils vous aident à manipuler et à changer des valeurs, à effectuer des calculs mathématiques, à comparer des valeurs, et bien plus encore.

Dans ce chapitre, vous allez découvrir comment ces opérateurs, expressions et instructions font leur travail dans JavaScript, et comment vous pouvez en tirer profit pour votre code.



N'oubliez pas de visiter le site Web compagnon de ce livre pour y télécharger les exemples de code de ce chapitre.

Exprimez-vous

Une *expression* est un morceau de code qui fournit une valeur. Les expressions peuvent servir à affecter une valeur à une variable, ou simplement avoir une valeur. Par exemple, les deux expressions suivantes sont parfaitement valides :

```
1 + 1
a = 1 ;
```

Les expressions peuvent être très courtes et très simples, comme dans l'exemple ci-dessus, ou être extrêmement complexes. Les morceaux de données qui servent à former une expression (comme le chiffre 1 ou la lettre a dans l'exemple précédent) sont appelés des *opérandes*.

Bonjour, opérateur

Un *opérateur*, c'est ce qui permet aux expressions de faire leur travail. Comme leur nom l'indique, les opérateurs opèrent sur les données pour fournir différents résultats. Dans les exemples ci-dessus, les signes + ou = sont des opérateurs.

Préséance des opérateurs

Une même expression contient bien souvent plusieurs opérateurs. Prenons un exemple :

```
a=1+2*3/4 ;
```

Selon l'ordre dans lequel les calculs sont effectués, la valeur finale pourrait être l'une des trois suivantes :

```
a = 1.75
a = 2.5
a = 2.25
```

Et la bonne réponse est… 2.5. Mais pourquoi ? Si nous demandons à plusieurs personnes d'effectuer ce calcul, l'une pourrait commencer par effectuer la division (3 / 4), une autre l'addition (1 + 2), et une troisième la multiplication (2 * 3).

Il doit donc y avoir une *bonne* manière de s'y prendre. Et c'est là où la *préséance des opérateurs* entre en scène. Celle-ci nous indique l'ordre dans lequel les opérateurs doivent être évalués dans une expression.

Les opérateurs sont divisés en groupes possédant différents niveaux de priorité, et qui sont numérotés de 0 (le plus prioritaire) à 18 (le moins prioritaire). C'est ce que décrit le Tableau 5.1.

Tableau 5.1 : La préséance des opérateurs.

Opérateur	Utilisation	Associativité de l'opérateur	Priorité	Exemple
(..)	regroupement	n/a	0 (priorité la plus élevée)	(1 + 3)
.....	accès aux propriétés de l'opérateur	de gauche à droite	1	maVoiture.color
[..]	accès à un tableau	de gauche à droite	1	chosesAFaire[4]
new... ()	crée un objet (avec une liste d'arguments)	n/a	1	new Voiture ("rouge")
function... ()	appel de fonction	de gauche à droite	2	function addNumbers (1,2)
new...	crée un objet (sans liste d'arguments)	de droite à gauche	2	new Voiture
... ++	incrément postfixé	n/a	3	nombre++
...--	décrément postfixé	n/a	3	nombre--
!...	non logique	de droite à gauche	4	! maValeur
~...	non au niveau du bit	de droite à gauche	4	~maValeur

- …	négation	de droite à gauche	4	- unNombre
++…	incrément préfixé	de droite à gauche	4	++unNombre
- -…	décrément préfixé	de droite à gauche	4	- -unNombre
typeof…	type de l'élément	de droite à gauche	4	typeof maVar
void…	vide	de droite à gauche	4	void (0)
delete…	suppression	de droite à gauche	4	delete objet. propriété
…*…	multiplication	de gauche à droite	5	résultat = 3 * 7
…/…	division	de gauche à droite	5	résultat = 3 / 7
… %…	reste de la division entière	de gauche à droite	5	résultat = 7 % 3
… +…	addition	de gauche à droite	6	résultat = 3 + 7
…-…	soustraction	de gauche à droite	6	résultat = 3 - 7
… <<…	décalage de bit à gauche	de gauche à droite	7	résultat = 3 << 7
…>>…	décalage de bit à droite	de gauche à droite	7	résultat = 3 >> 7
…>>>…	décalage de bit à droite (non signé)	de gauche à droite	7	résultat = 3 >>> 7
… <…	plus petit que	de gauche à droite	8	a<b
… <=…	plus petit que ou égal à	de gauche à droite	8	a <= b
…>…	plus grand que	de gauche à droite	8	a>b
…>=…	plus grand que ou égal à	de gauche à droite	8	a >= b
… in…	dans	de gauche à droite	8	valeur in valeurs
… instanceof. .	instance de	de gauche à droite	8	maVoiture instanceof Voiture
… ==…	égalité	de gauche à droite	9	3 == "3" // true
…! =…	inégalité	de gauche à droite	9	3 ! = "3" // false
… ===…	égalité stricte	de gauche à droite	9	3 === "3" // false
…! ==…	inégalité stricte	de gauche à droite	9	3 ! == "3" // true
… &…	et (and) au niveau du bit	de gauche à droite	10	resultat = a & b
… ^…	ou exclusif (xor) au niveau du bit	de gauche à droite	11	resultat = a ^ b

... |...	ou (or) au niveau du bit	de gauche à droite	12	resultat = a | b
... & &...	et (and) logique	de gauche à droite	13	a & & b
... ||...	ou (or) logique	de gauche à droite	14	a || b
...?... : ...	conditionnel	de droite à gauche	15	a ?3 : 7
... =...	affectation	de droite à gauche	16	a=3
... +=...	affectation	de droite à gauche	16	a += 3
...-=...	affectation	de droite à gauche	16	a -= 3
...*=...	affection	de droite à gauche	16	a *= 3
.../=...	affectation	de droite à gauche	16	a /= 3
... % =...	affectation	de droite à gauche	16	a % = 3
... <<=...	affectation	de droite à gauche	16	a <<= 3
...>>=...	affectation	de droite à gauche	16	a >>= 3
...>>>=...	affectation	de droite à gauche	16	a >>>= 3
... & =...	affectation	de droite à gauche	16	a & = 3
... ^=...	affectation	de droite à gauche	16	a ^= 3
... |=...	affectation	de droite à gauche	16	a |= 3
yield...	expression renvoyant un objet itérable	de droite à gauche	17	yield [expression]
..., ...	virgule / séquence	de gauche à droite	18	a + b, c + d



L'opérateur qui porte le plus petit numéro est dit avoir la plus haute priorité. Cela peut sembler un peu étrange, mais si vous vous figurez être la première personne d'une file d'attente (celle qui a le rang 0 en l'occurrence), et donc que vous allez être le premier à recevoir votre sandwich et votre café, vous comprendrez mieux ce qui se passe.

Lorsqu'une expression contient deux opérateurs, ou plus, ayant la même priorité, ceux-ci sont évalués selon leur *associativité*. Cette associativité détermine si les opérateurs sont évalués en allant de la gauche vers la droite, ou de la droite vers la gauche.

Utiliser des parenthèses

Les parenthèses représentent l'opérateur qui a le plus haut niveau de préséance, ou de priorité. Dans la plupart des cas, vous pouvez ignorer les règles décrites dans le Tableau 5.1 en regroupant simplement les opérations à l'aide de parenthèses. Par exemple, l'expression proposée vers le début de ce chapitre (a + 1 + 2 * 3 / 4 ;) pourrait être clarifiée de différentes manières :

```
a = (1 + 2) * (3 / 4); // résultat : 2.25
a = (1 + (2 * 3)) / 4; // résultat: 1.75
a = ((1 + 2) *3) / 4; // résultat: 2.25
```

```
a = 1 + ((2 * 3) / 4); // résultat: 2.5
```

Les parenthèses utilisées dans une expression forcent JavaScript à évaluer en premier leur contenu, et ce en allant de l'intérieur vers l'extérieur. Les autres opérations, celles qui sont situées en dehors de ces parenthèses, ne sont évaluées qu'ensuite. Prenons un petit exemple :

```
a = 1 + ((2 * 3) / 4) ;
```

Ici, la priorité des opérateurs est tout à fait explicite. La multiplication (parenthèses intérieures) est effectuée en premier, soit :

```
a = 1 + (6 / 4) ;
```

JavaScript évalue maintenant le contenu des parenthèses qui restent, comme ceci :

```
a = 1 + 1.5 ;
```

Et il ne lui reste plus qu'à effectuer l'addition :

```
a = 2.5 ;
```

Types d'opérateurs

JavaScript possède de nombreux types d'opérateurs. Nous allons nous intéresser ici à ceux qui sont le plus utilisés.

Opérateurs d'affectation

Un *opérateur d'affectation*, comme son nom l'indique, affecte la valeur de l'opérande placé à droite à l'opérande de gauche. Exemple :

```
a=5 ;
```

Une fois cette expression exécutée, la variable `a` possède comme valeur 5. Mais vous pouvez également enchaîner plusieurs opérateurs d'affectation afin de définir un contenu identique pour plusieurs variables dans une même instruction, comme dans cet exemple :

```
a=b=c=5 ;
```

Du fait que l'associativité de cet opérateur s'établit de la droite vers la gauche, 5 est d'abord assigné à la variable `c`, puis la valeur de `c` est recopiée dans `b`, et enfin le contenu de `b` est affecté à `a`. Les trois variables `a`, `b` et `c` possèdent donc toutes la même valeur, 5.

Essayez maintenant de deviner la valeur finale de `a` une fois ces expressions évaluées :

```
var b = 1;
var a = b += c = 5;
```

Si vous ne trouvez pas la réponse, ouvrez la console JavaScript de Chrome et tapez ces lignes sans oublier de les valider en appuyant sur la touche Entrée. Le résultat indique que `a` est égal à 6.



Vous trouverez une liste complète des différents opérateurs d'affectation dans la section « Combiner des opérateurs », plus loin dans ce chapitre.

Opérateurs de comparaison

Les *opérateurs de comparaison* testent l'égalité, ou la non-égalité, entre les opérandes. Ils retournent ensuite une valeur `true` (vrai) ou `false` (faux).

Le Tableau 5.2 liste les opérateurs de comparaison de JavaScript.

Tableau 5.2 : Les opérateurs de comparaison de JavaScript.

Opérateur	Description	Exemple
==	Égalité	3 == "3" // true
! =	Inégalité	3 ! = 3 // false
===	Égalité stricte	3 === "3" // false
! ==	Inégalité stricte	3 ! == "3" // true
>	Plus grand que	7 > 1 // true
>=	Plus grand que ou égal à	7 >= 7 // true
<	Plus petit que	7 < 10 // true
<=	Plus petit que ou égal à	2 <= 2 // true

Opérateurs arithmétiques

Les *opérateurs arithmétiques* effectuent des calculs mathématiques sur leurs opérandes et en retournent le résultat. Le Tableau 5.3 fournit une liste complète des opérateurs arithmétiques de JavaScript.

Tableau 5.3 : Les opérateurs arithmétiques de JavaScript.

Opérateur	Description	Exemple
+	Addition	a=1+1
-	Soustraction	a = 10 - 1
*	Multiplication	a=2*2
/	Division	a=8/2
%	Modulo	a=5 % 2
++	Incrément	a = ++b
a = b++		
a++		
- -	Décrément	a = --b
a = b-		
a--		

Le Listing 5.1 montre des opérateurs arithmétiques au travail.

LISTING 5.1 : Utiliser des opérateurs arithmétiques.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
  <title>Opérateurs arithmétiques</title>
</head>
<body>
  <h1>Le Jeu de l'Anniversaire Dévoilé</h1>
  <p>
  <ul>
  <li>Entrez le nombre 7</li>
  <li>Multipliez par le jour de votre naissance</li>
  <li>Soustrayez 1</li>
  <li>Multipliez par 13</li>
  <li>Ajoutez le mois de votre naissance</li>
  <li>Ajoutez 3</li>
  <li>Multipliez par 11</li>
  <li>Soustrayez le jour de votre naissance</li>
  <li>Soustrayez le mois de votre naissance</li>
  <li>Divisez par 10</li>
  <li>Ajoutez 11</li>
  <li>Divisez par 100</li>
  </ul>
  </p>
  <script>
  var numberSeven = Number(prompt(‘Entrez le nombre 7'));
  var birthDay = Number(prompt(‘Entrez le jour de votre naissance'));
  var calculation = numberSeven * birthDay;
  calculation = calculation - 1;
  calculation = calculation * 13;
  var birthMonth = Number(prompt(‘Entrez le mois de votre naissance'));
  calculation = calculation + birthMonth;
  calculation = calculation + 3;
  calculation = calculation * 11
  calculation = calculation - birthDay;
  calculation = calculation - birthMonth;
  calculation = calculation / 10;
  calculation = calculation + 11;
  calculation = calculation / 100;

  document.write("Voici votre date d'anniversaire : " + calculation);
 </script>
</body>
</html>
```

Le résultat de l'exécution de ce programme est illustré sur la Figure 5.1.

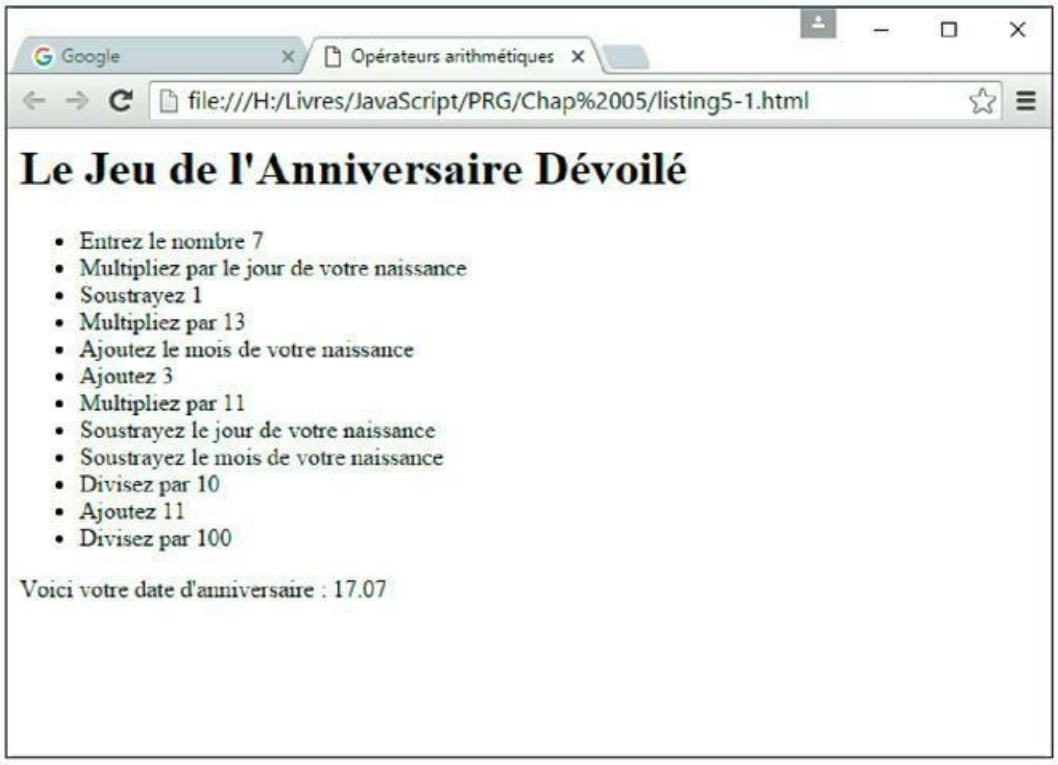


FIGURE 5.1 : JavaScript est capable de trouver la date de votre anniversaire !

Opérateur chaîne

L'opérateur chaîne effectue des opérations en utilisant deux chaînes de caractères. Lorsqu'il est utilisé avec des chaînes, l'opérateur + devient l'*opérateur de concaténation*. Il sert alors à fusionner deux chaînes en une seule, en les mettant bout à bout. Notez que, dans ce cas, aucune espace n'est ajoutée. Il est donc très courant de voir des instructions comme celle qui suit, où des chaînes ne contenant rien d'autre qu'une espace blanche sont concaténées avec d'autres de manière à effectuer une séparation pour que la phrase soit cohérente :

```
var salut = "Bonjour, " + Prenom + ". Je suis" + " " + Humeur + " de vous
voir.";
```

Si la variable Prénom contient par exemple *Pierre*, et la variable `Humeur` le mot *heureux*, la chaîne résultante, `salut`, aura pour valeur *Bonjour, Pierre. Je suis heureux de vous voir*.

Opérateurs au niveau du bit

Les *opérateurs au niveau du bit* traitent les opérandes comme des représentations binaires signées sur 32 bits dans le format dit de complément à 2. Voyons ce que cela signifie, en commençant par le terme *binaire*.

Les nombres binaires sont des suites de 1 et de 0, en fait la seule chose que soit réellement capable de comprendre un ordinateur. Voici par exemple comment écrire le chiffre 1 en numérotation binaire sur 32 bits :

```
00000000000000000000000000000001
```

La position la plus à droite a une valeur de 1. Chaque décalage de la droite vers la gauche se traduit par un doublement de la position précédente (une succession de puissances de 2). Par exemple :

```
00000000000000000000000000000101
```

est la représentation binaire sur 32 bits du chiffre 5 (soit 1*2^2 + 0*2^1 + 1*2^0).

La notion d'*entier signé* signifie que les nombres entiers peuvent être représentés de cette manière, qu'ils soient positifs ou négatifs.

Cela est encore précisé par l'expression *complément à 2*. Dans ce cas, l'opposé de toute valeur binaire positive est son équivalent négatif (et réciproquement bien entendu). Pour changer la valeur 5 en -5, il suffit d'inverser les bits, comme ceci :

```
11111111111111111111111111111011
```

Les opérateurs au niveau du bit convertissent les nombres dans ce format binaire sur 32 bits, puis ils les convertissent en sens inverse, c'est-à-dire en nombre que nous pourrions considérer comme normaux une fois l'opération effectuée.



Ces opérateurs sont, il faut le reconnaître, plutôt difficiles à comprendre pour le débutant. De plus, ils sont rarement employés en JavaScript. Mais nous ferions preuve de négligence si nous ne les abordions pas.

Le Tableau 5.4 liste les opérateurs au niveau du bit de JavaScript.

Tableau 5.4 : Les opérateurs au niveau du bit de JavaScript.

Opérateur binaire	Utilisation	Description
AND	a & b	Retourne un 1 pour chaque position où les bits correspondants des deux opérandes sont égaux à 1
OR	a | b	Retourne un 1 pour chaque position où les bits correspondants de l'un au moins des deux opérandes sont égaux à 1
XOR	a^b	Retourne un 1 pour chaque position où les bits correspondants de l'un des deux opérandes, mais pas des deux, sont égaux à 1
NOT	~a	Inverse les bits de son opérande
Décalage à gauche	a << b	Décale a en représentation binaire de b (<32) bits vers la gauche, les bits perdus sur la droite étant remplacés par des 0
Décalage à droite avec propagation de signe	a >> b	Décale a en représentation binaire de b (<32) bits vers la droite, les bits perdus étant éliminés
Décalage à droite avec remplissage par des 0	a >>> b	Décale a en représentation binaire de b (<32 bits) vers la droite, les bits perdus sur la gauche étant remplacés par des 0

La Figure 5.2 illustre le comportement de ces opérateurs dans la console JavaScript de Chrome.

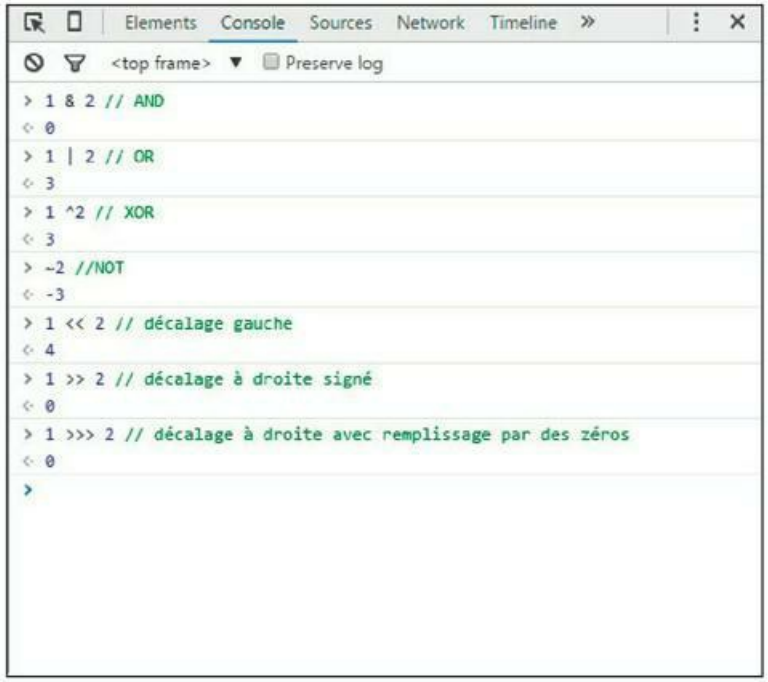


FIGURE 5.2 : Les opérateurs au niveau du bit de JavaScript.

Opérateurs logiques

Les *opérateurs logiques* évaluent une expression (également logique !) comme étant vraie ou fausse. Il en existe trois, qui sont décrits dans le Tableau 5.5.

Tableau 5.5 : Opérateurs logiques.

Opérateur	Signification	Description
& &	And (et)	Retourne `true` (vrai) si les deux opérandes sont `true`. Sinon, retourne `false`.
||	Or (ou)	Retourne `false` (faux) si les deux opérandes sont `false`. Sinon, retourne `true`.
!	Not (non)	Inverse la valeur logique d'un unique opérande.

Opérateurs spéciaux

Les opérateurs spéciaux de JavaScript sont un mix d'autres symboles et mots qui exécutent des fonctions diverses et importantes.

Opérateur conditionnel

L'*opérateur conditionnel* (également appelé *opérateur ternaire*) utilise trois opérandes. Il évalue une valeur logique, puis retourne une valeur basée sur le résultat vrai ou faux de l'expression. Il s'agit du seul opérateur qui nécessite trois opérandes. Par exemple :

```
var estPlusGrandQueDix = (valeur > 10) ? "plus grand que 10" :
      "pas plus grand que 10";
```

Opérateur virgule

L'*opérateur virgule* évalue deux opérandes et retourne la valeur du second. Il est le plus souvent utilisé pour réaliser des affectations multiples, ou encore d'autres opérations à l'intérieur de boucles. Il peut également servir de raccourci pour initialiser des variables. Par exemple :

```
var a = 10 , b = 0 ;
```

Du fait que la virgule a le plus bas niveau de priorité parmi tous les opérateurs, ses opérandes sont toujours évalués séparément.

Opérateur de suppression

L'*opérateur de suppression* (`delete`) retire une propriété d'un objet, ou un élément d'un tableau.



Lorsque vous utilisez cet opérateur pour supprimer un élément d'un tableau, la longueur de ce tableau ne change pas. L'élément qui a été enlevé prendra simplement la valeur `undefined` (non défini).

Prenons un exemple :

```
var animaux = ["chien","chat","oiseau","pieuvre"];
console.log (animaux[3]); // retourne "pieuvre"
delete animaux[3];
console.log (animaux[3]); // retourne "undefined"
```

Opérateur in

L'*opérateur* `in` (dans) retourne vrai (`true`) si la valeur spécifiée existe dans un tableau ou dans un objet. Par exemple :

```
var animaux = ["chien","chat","oiseau","pieuvre"];
if (3 in animaux) {
  console.log ("il est bien là !");
}
```

Dans cet exemple, si le tableau `animaux` possède un élément dont l'indice vaut 3, le code affichera un message pour dire qu'il est bien là.

Opérateur instanceof

L'*opérateur* ***`instanceof`*** retourne `true` si l'objet que vous spécifiez est bien du type demandé. Par exemple :

```
var mChaine = new String();
if (maChaine instanceof String) {
  console.log("Mais oui, c'est une chaîne !");
}
```

Opérateur new

L'*opérateur* `new` crée une instance d'un objet. Comme vous le verrez dans le Chapitre 8, JavaScript possède plusieurs types d'objets intégrés, mais vous pouvez aussi définir les vôtres. Dans l'exemple qui suit, `Date ()` est un objet JavaScript, tandis que `Animal ()` et `Fleur ()` sont des objets créés par le programmeur afin de répondre à des besoins particuliers.

```
var aujourdhui = new Date();
var oiseau = new Animal();
var marguerite = new Fleur();
```

Opérateur this

L'*opérateur* `this` fait référence à l'objet courant. Il est fréquemment utilisé pour retrouver les propriétés d'un objet sans avoir à saisir le nom de celui-ci.

Le Chapitre 8 traite plus en détail de l'opérateur `this`.

Opérateur typeof

L'*opérateur* `typeof` retourne une chaîne contenant le type de l'opérande :

```
var maPetiteEntreprise = "Chez Milou, on trouve tout !";
console.log typeof maPetiteEntreprise; // retourne "string"
```

Opérateur void

L'*opérateur* `void` (vide) évalue l'expression dans l'opérande sans retourner de valeur. L'usage le plus fréquent de cet opérateur se voit dans des documents HTML lorsqu'un lien est nécessaire, mais que le créateur de ce lien veut en annuler le comportement par défaut *via* JavaScript. Par exemple :

```
<a href="javascript : void (0) ; ">C'est un lien, mais il ne fait rien</ a>
```

Combiner les opérateurs

Vous pouvez combiner les opérateurs d'affectation avec d'autres, ce qui permet de créer un raccourci lors de l'affectation du résultat à une variable. Par exemple, les deux instructions suivantes donnent le même résultat :

```
a = a + 10;
a += 10;
```

Le Tableau 5.6 liste toutes les combinaisons possibles des opérateurs d'affectation avec d'autres opérateurs.

Tableau 5.6 : Combiner les opérateurs d'affectation avec d'autres opérateurs.

Nom	Raccourci	Opérateur standard
Affectation	x=y	x=y
Affectation avec addition	x += y	x=x+y

Affectation avec soustraction	x -= y	x=x-y
Affectation avec multiplication	x *= y	x=x*y
Affectation avec division	x /= y	x=x/y
Affectation avec reste entier	x % = y	x=x % y
Affectation avec décalage gauche	x <<= y	x = x << y
Affectation avec décalage droit	x >>= y	x = x >> y
Affectation avec décalage droit non signé	x >>>= y	x = x <<< y
Affectation avec AND (au niveau du bit)	x & = y	x=x & y
Affectation avec XOR (au niveau du bit)	x ^= y	x=x^y
Affectation avec OR (au niveau du bit)	x |= y	x=x|y

Chapitre 6

Rester dans le flux avec les boucles et les branchements

DANS CE CHAPITRE :

- **Découvrir les branchements if/else**
- **Comprendre les différents types de boucles**
- **Utiliser des boucles pour répéter des instructions**
- **Parcourir les valeurs d'un tableau**

« Il n'est pas difficile de prendre des décisions quand on en connaît les valeurs. »

Roy Disney

Dans les chapitres précédents de ce livre, nous nous sommes intéressés au code « linaire » de JavaScript. Cependant, et plus souvent que vous ne le souhaiteriez sans doute, vous allez avoir besoin de faire des choix, ou encore de répéter des séries d'instructions un certain nombre de fois avec des valeurs différentes. C'est pourquoi nous allons parler ici de boucles et de branchements.



N'oubliez pas de visiter le site Web compagnon de ce livre pour y télécharger les exemples de ce chapitre !

Soyez branché !

Les instructions de *boucle* et de *branchement* sont appelées aussi *instructions de contrôle* parce qu'elles permettent de contrôler l'ordre dans lequel JavaScript exécute ses programmes. Vous pouvez utiliser des instructions de branchement pour décider des voies à emprunter dans votre code JavaScript. Les *boucles* sont la méthode la plus simple pour regrouper des instructions JavaScript dans un programme.

La logique des programmes JavaScript réside souvent en un point où un certain choix doit être effectué, ce qui fait ensuite toute la différence. La Figure 6.1 illustre ce genre de décision.



FIGURE 6.1 : Dans un embranchement, il faut choisir la direction à prendre (photo de Daniel Oines).

If... else

Les instructions `if` et `else` travaillent conjointement pour évaluer une expression logique et exécuter différentes instructions en fonction du résultat. Bien souvent, l'instruction `if` est exécutée seule. Par contre, `else` ne peut pas exister sans son `if`.

La syntaxe de base est la suivante :

```
if (condition) {
...
}
```

La condition est une expression quelconque qui peut être évaluée comme étant une valeur booléenne (donc vraie ou fausse). Si le résultat de cette expression est `true` (vrai), alors les instructions comprises entre les accolades sont exécutées. Dans le cas contraire, elles sont simplement ignorées.

L'instruction `else` intervient dans le cas où le `if` est évalué comme étant de valeur `false`. Par exemple :

```
var age = 19;
if (age < 24){
  document.write ("Vous êtes trop jeune pour être sénateur en France !");
  } else {
  document.write ("Présentez-vous aux élections. Vous avez peut-être votre
chance.");
}
```

De nombreux autres langages de programmation possèdent une instruction appelée `elseif`, qui peut être utilisée de multiples fois à l'intérieur d'une instruction `if... else` jusqu'à ce que le programme retourne une valeur `true`. Mais ce n'est pas le cas de JavaScript.

Cependant, vous pouvez reproduire la même fonctionnalité en plaçant un espace entre le `if` et le `else`. En voici un exemple :

```
if (heure < 12){
  document.write("Bon matin !");
} else if (heure < 17){
  document.write("Bon après-midi !");
} else if (heure < 20){
  document.write("Bonne soirée !");
} else {
  document.write("Bonne nuit !");
}
```

Notez ici l'emploi des sauts de ligne et des espaces. La plupart des gens ont leur propre style pour écrire des instructions `if… else`. Par exemple, il est possible de rencontrer moins de sauts de ligne ou d'espaces entre les mots-clés et les accolades. Cela est parfaitement valide. Cependant, il est en général préférable de bien distinguer tout cela, en particulier pour des raisons de lisibilité et de compréhension.

L'INSTRUCTION IF… ELSE EN RACCOURCI

L'instruction `if… else` peut être raccourcie. La première méthode consiste à utiliser un opérateur ternaire à la place du `if... else`. Cela rend cependant le code un peu plus difficile à lire, comme dans :

```
var formuleDePolitesse = (heure <
12 ?
       "Bon matin  !"  :
      "Bonjour.");
```

Dans ce cas, la valeur de formuleDePolitesse est égale à "Bon matin !" avant midi, et à "Bonjour." sinon.

Une autre technique consiste à faire appel à l'opérateur & & (AND, ou *et* logique). Rappelez-vous que le second opérande ne sera évalué que si le premier est vrai. Les programmeurs appellent souvent cela un court-circuit, car le second opérande n'a pas besoin d'être évalué si le premier renvoie `false`.

```
heure < 12 && document.write("Bon
matin  !");
```

Dans cet exemple, l'instruction & & regarde si l'heure courante est inférieure à midi. Dans ce cas, elle affiche "Bon matin !". Sinon, il ne va rien se passer, du fait de l'effet de bord de l'opérateur & &.

Cette méthode n'est pas courante, en particulier parce qu'elle n'est pas simple à comprendre, et donc source de confusion. Cependant, elle pourrait peut-être vous rendre service un jour, et mérite donc à ce titre d'être connue.

Switch

L'instruction `switch` fait un choix parmi une série de lignes de code en fonction de la valeur d'une certaine expression. Chaque valeur dans une instruction `switch` est appelée une *valise* (ou `case`, que l'on pourrait aussi traduire par coffret, ou tout autre contenant du même genre). Disons un *cas* pour faire simple (on dit aussi une *clause* `case`)…

La syntaxe de cette instruction se présente ainsi :

```
switch (expression) {
  case valeur1:
  // Instructions
  break;
  case valeur2:
  // Instructions
  break;
  case valeur3:
  // Instructions
  break;
  default:
  // Instructions
  break;
}
```

Notez en particulier l'emploi de l'instruction `break` à la suite de chaque cas. Elle demande à l'instruction `switch` de s'interrompre et de terminer son exécution. Sans cela, `switch` continuerait à s'exécuter, et évaluerait donc toutes les expressions qui suivent, sans savoir si l'expression vérifie encore la condition posée.



L'oubli d'une instruction `break` dans une structure `switch` peut engendrer de gros problèmes. N'oubliez donc jamais de la spécifier. Du fait que l'instruction `switch` exécute tout ce qui suit une fois que la condition initiale a été évaluée comme étant vraie (`true`) les résultats produits par le code pourraient bien devenir totalement imprévisibles si vous oubliez une seule instruction `break`. Ces problèmes sont généralement difficiles à identifier, car ils ne produisent pas forcément des erreurs détectables, mais plutôt des résultats incorrects.

Si aucune des clauses `case` n'est satisfaite, l'instruction `switch` va rechercher une clause `default` et exécuter les instructions qu'elle contient.



La clause `default` est l'exception à la règle qui dit que vous devriez toujours utiliser une instruction `break` entre les clauses `case`. Du fait que `default` est (normalement) ce qui termine l'instruction `switch`, l'exécution de celle-ci se terminera mécaniquement une fois la clause `default` rencontrée.

Le Listing 6.1 vous montre un exemple d'utilisation de l'instruction `switch`.

LISTING 6.1 : Utilisation d'une instruction switch pour personnaliser un texte de salutation.

```
var languagePreference = "Espagnol"
switch (languagePreference){
 case "Anglais":
  console.log("Hello!");
  break;
 case "Espagnol":
  console.log("Hola!");
  break;
 case "Allemand":
  console.log("Guten Tag!");
  break;
 case "Français":
  console.log("Bonjour!");
  break;
 default:
  console.log("Désolé, je ne parle pas " + languagePreference + "!");
}
```

Boucler les boucles

Une *boucle* exécute les mêmes instructions un certain nombre de fois. JavaScript propose plusieurs types de boucles :

- `for`
- `for…in`
- `do… while`
- `while`

La boucle for

L'instruction `for` (pour) produit une boucle en utilisant trois expressions :

- **Initialisation :** La valeur initiale de la variable servant de *compteur* de boucle.
- **Condition :** Une expression booléenne qui doit être évaluée à chaque itération de la boucle pour déterminer si celle-ci doit ou non se poursuivre.
- **Expression finale :** Une expression qui doit être évaluée après chaque itération.

Même si la présence de ces trois expressions n'est pas une obligation statutaire, elles sont virtuellement toujours présentes. Les boucles `for` sont généralement utilisées pour exécuter du code un nombre prédéterminé de fois. Prenons un exemple simple :

```
for (var x = 1; x < 10; x++){
```

```
  console.log(x);
}
```

Détaillons ces instructions :

1. **Une nouvelle variable, en l'occurrence** `x`**, est initialisée avec une valeur égale à 1 (la graine est semée…).**
2. **Un test est effectué afin de déterminer si** `x` **est plus petit que 10.**
3. **Si la condition est vérifiée, les instructions contenues dans la boucle sont exécutées (ici, il s'agit simplement d'une instruction** `console.log` **qui se contente d'afficher la valeur courante de** `x`**).**
4. **Une fois les instructions exécutées, la valeur de** `x` **est incrémentée d'une unité via l'opérateur ++.**
5. **La condition est à nouveau testée. Si elle reste vérifiée, le code « boucle » en se rebranchant à l'Étape 3.**
6. **Si la condition n'est plus vérifiée (**`x` **vaut donc 10 dans cet exemple), la boucle se termine.**

La Figure 6.2 illustre l'exécution de ce code dans la console JavaScript de Chrome.

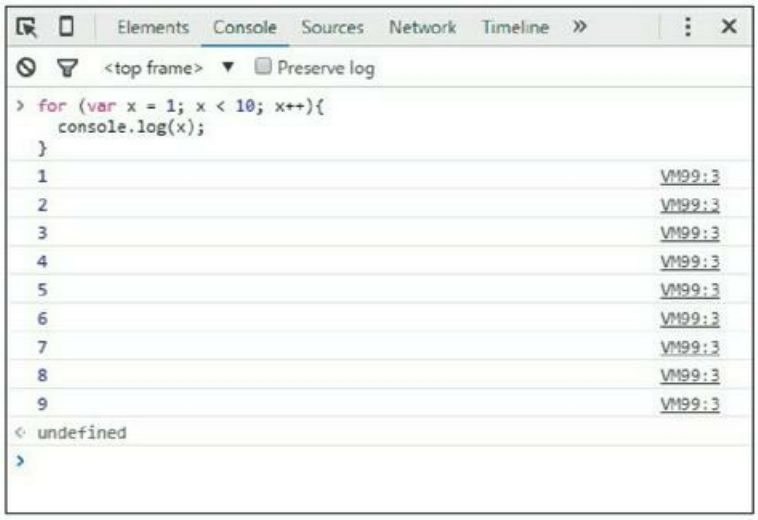


FIGURE 6.2 : Une boucle qui compte de 1 à 9.

Parcourir un tableau

Vous pouvez utiliser des boucles `for` pour lister le contenu d'un tableau en comparant la valeur du compteur avec la propriété `length` du tableau. N'oubliez pas cependant que, dans un tableau JavaScript, le premier indice est 0, et donc que la propriété `array.length` est supérieure d'une unité au dernier indice du tableau. C'est pourquoi nous avons ajouté -1 dans le Listing 6.2.

LISTING 6.2 : Lister le contenu d'un tableau avec une boucle for.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
  <title>Tableau de codes de zone</title>
</head>
<body>
  <script>
    var areaCodes = ["770", "404", "718", "202", "901", "305", "312",
"313",
"215", "803"];
    for (x=0; x < areaCodes.length - 1; x++){
      document.write("Différents codes de zone : " + areaCodes[x] + "
<br>");
    }
  </script>
</body>
</html>
```

La Figure 6.3 illustre l'exécution du code du Listing 6.2.

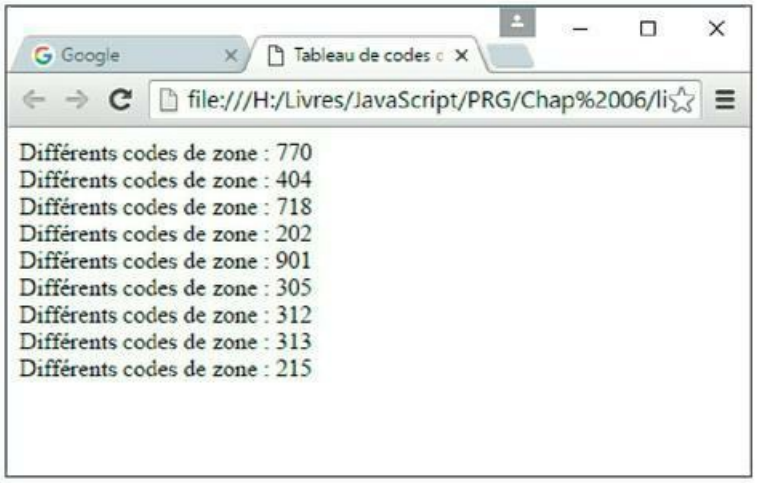

Google
Tableau de codes
file:///H:/Livres/JavaScript/PRG/Chap%2006/li
Différents codes de zone : 770
Différents codes de zone : 404
Différents codes de zone : 718
Différents codes de zone : 202
Différents codes de zone : 901
Différents codes de zone : 305
Différents codes de zone : 312
Différents codes de zone : 313
Différents codes de zone : 215

FIGURE 6.3 : Parcourir le contenu d'un tableau avec une boucle for.

La boucle for...in

Les instructions `for...in` servent notamment à parcourir les propriétés d'un objet. Mais vous pouvez également y faire appel pour parcourir les valeurs d'un tableau.



La boucle `for...in` a son talon d'Achille. Elle ne tient en effet pas compte de l'ordre des propriétés ou des éléments qu'elle parcourt. Pour cette raison, mais aussi parce que l'exécution de `for...in` est lente, il est nettement préférable de s'en tenir à une boucle `for` pour balayer la contenu d'un tableau.

Les objets sont des conteneurs de données qui possèdent des propriétés (ce que sont ces données) et des méthodes (ce qu'elles font). Les navigateurs Web possèdent un ensemble d'objets intégrés que les programmeurs peuvent

utiliser pour contrôler leur fonctionnement. Le plus basique de tous est l'objet `Document`. La méthode `write` de cet objet, par exemple, demande à votre navigateur d'insérer la valeur spécifiée dans le document HTML.

L'objet `Document` possède également des propriétés dont il se sert pour obtenir des informations sur le document courant, et éventuellement les retourner aux programmeurs. Par exemple, la collection `Document.images` contient toutes les balises `img` du document HTML courant.

Dans le Listing 6.3, une boucle `for…in` permet de lister toutes les propriétés de l'objet `Document`.

LISTING 6.3 : Parcourir les propriétés de l'objet Document avec une boucle for…in.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
  <title>Propriétés de l'objet Document</title>
  <style>
    .columns {
    -webkit-column-count: 6; // Chrome, Safari, Opera
    -moz-column-count: 6; // Firefox
    column-count:  6;
    }
  </style>
</head>
<body>
  <div class="columns">
  <script>
    for (var prop in document){
      document.write (prop + "<br>");
    }
  </script>
  </div>
</body>
</html>
```

Les résultats produits par ce code sont illustrés sur la Figure 6.4.

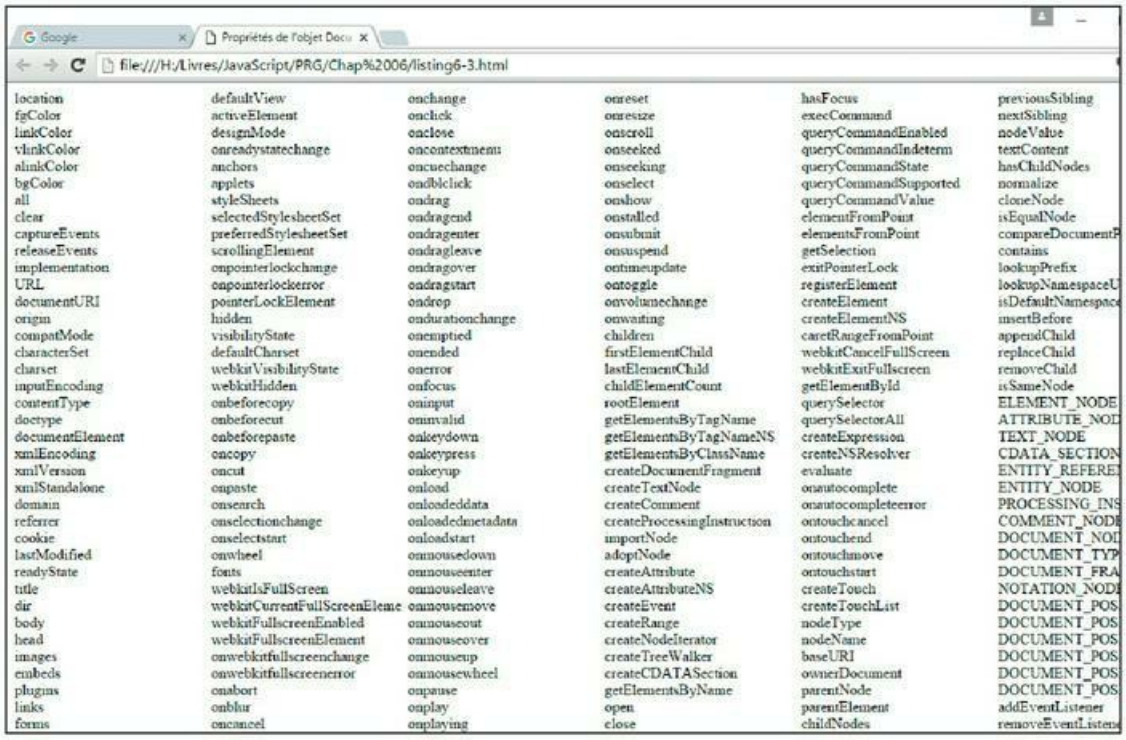

Google
Propriétés de l'objet Docu
file:///H:/Livres/JavaScript/PRG/Chap%2006/listing6-3.html
location defaultView onchange onreset hasFocus previousSibling
fgColor activeElement onclick onresize execCommand nextSibling
linkColor designMode onclose onscroll queryCommandEnabled nodeValue
vlinkColor onreadystatechange oncontextmenu onseeked queryCommandIndeterm textContent
alinkColor anchors oncuechange onseeking queryCommandState hasChildNodes
bgColor applets ondblclick onselect queryCommandSupported normalize
all styleSheets ondrag onshow queryCommandValue cloneNode
clear selectedStylesheetSet ondragend onstalled elementFromPoint isEqualNode
captureEvents preferredStylesheetSet ondragenter onsubmit elementsFromPoint compareDocumentP
releaseEvents scrollingElement ondragleave onsuspend getSelection contains
implementation onpointerlockchange ondragover ontimeupdate exitPointerLock lookupPrefix
URL onpointerlockerror ondragstart ontoggle registerElement lookupNamespaceU
documentURI pointerLockElement ondrop onvolumechange createElement isDefaultNamespace
origin hidden ondurationchange onwaiting createElementNS insertBefore
compatMode visibilityState onemptied children caretRangeFromPoint appendChild
characterSet defaultCharset onended firstElementChild webkitCancelFullScreen replaceChild
charset webkitVisibilityState onerror lastElementChild webkitExitFullscreen removeChild
inputEncoding webkitHidden onfocus childElementCount getElementById isSameNode
contentType onbeforecopy oninput rootElement querySelector ELEMENT_NODE
doctype onbeforecut oninvalid getElementsByTagName querySelectorAll ATTRIBUTE_NOD
documentElement onbeforepaste onkeydown getElementsByTagNameNS createExpression TEXT_NODE
xmlEncoding oncopy onkeypress getElementsByClassName createNSResolver CDATA_SECTION
xmlVersion oncut onkeyup createDocumentFragment evaluate ENTITY_REFERE
xmlStandalone onpaste onload createTextNode onautocomplete ENTITY_NODE
domain onsearch onloadeddata createComment onautocompleteerror PROCESSING_INS
referrer onselectionchange onloadedmetadata createProcessingInstruction ontouchcancel COMMENT_NOD
cookie onselectstart onloadstart importNode ontouchend DOCUMENT_NOD
lastModified onwheel onmousedown adoptNode ontouchmove DOCUMENT_TYP
readyState fonts onmouseenter createAttribute ontouchstart DOCUMENT_FRA
title webkitIsFullScreen onmouseleave createAttributeNS createTouch NOTATION_NOD
dir webkitCurrentFullScreenEleme onmousemove createEvent createTouchList DOCUMENT_POS
body webkitFullscreenEnabled onmouseout createRange nodeType DOCUMENT_POS
head webkitFullscreenElement onmouseover createNodeIterator nodeName DOCUMENT_POS
images onwebkitfullscreenchange onmouseup createTreeWalker baseURI DOCUMENT_POS
embeds onwebkitfullscreenerror onmousewheel createCDATASection ownerDocument DOCUMENT_POS
plugins onabort onpause getElementsByName parentNode DOCUMENT_POS
links onblur onplay open parentElement addEventListener
forms oncancel onplaying close childNodes removeEventListen

FIGURE 6.4 : Une boucle for…infor… in permet de lister toutes les propriétés de l'objet Document.

Vous pouvez aussi utiliser une boucle `for…in` pour retrouver les valeurs contenues dans les propriétés d'un objet, et pas seulement le nom des celles-ci. C'est ce que vous propose le code du Listing 6.4.

LISTING 6.4 : Afficher les noms des propriétés de l'objet Document et leur valeur à l'aide d'une boucle for...in.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
  <title>Propriétés de l'objet Document avec valeurs</title>
  <style>
    .columns{
      -webkit-column-count: 6; /* Chrome, Safari, Opera */
      -moz-column-count: 6; /* Firefox */
       column-count:  6;
    }
  </style>
  </head>
  <body>
    <div class="columns">
      <script>
        for (var prop in document){
          document.write (prop + ": " + document[prop] + "<br>");
        }
      </script>
    </div>
 </body>
</html>
```

La Figure 6.5 illustre la sortie produite par ce programme. Vous pourrez remarquer que de nombreuses valeurs sont placées entre des crochets droits ([]). Ceux-ci indiquent que la valeur de la propriété contient de multiples éléments, comme dans le cas d'un tableau ou d'un objet.

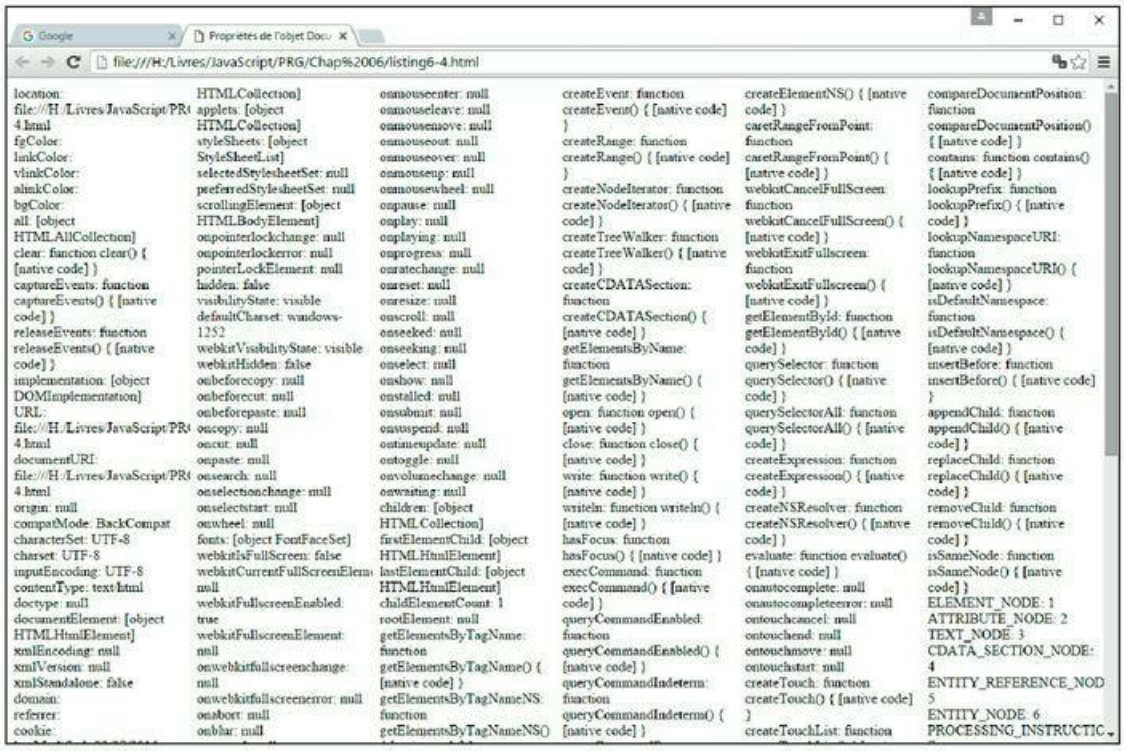


FIGURE 6.5 : Afficher les noms des propriétés de l'objet Document et leur valeur à l'aide d'une boucle for...in.

Boucles while

L'instruction `while` (tant que) produit une boucle qui est exécutée tant qu'une certaine condition reste vérifiée (`true`). Le Listing 6.5 propose un exemple de page Web contenant une boucle `while`.

LISTING 6.5 : La boucle while.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
  <title>Trouvez le mot !</title>
</head>
<body>
  <script>
    var guessedWord = prompt("Quel est le mot auquel je pense ?");
    while (guessedWord != "sandwich") { // tant que le mot attendu n'est
pas
        sandwich
      prompt("Non. C'est faux. Essayez encore.");
    }
      alert("Félicitations ! Vous avez trouvé !"); // après cela, la
boucle se
        termine
  </script>
</body>
</html>
```

Boucles do... while

La boucle `do... while` est semblable à `while`, si ce n'est que les instructions sont exécutées avant que l'expression de condition ne soit testée. La conséquence est qu'une boucle `do... while` est exécutée au moins une fois, tandis qu'une boucle `while` peut très bien ne jamais l'être si la condition est fausse dès le départ.

Le Listing 6.6 illustre l'emploi d'une boucle do… while.

LISTING 6.6 : La boucle do... while.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
 <title>Comptons !</title>
</head>
<body>
  <script>
    var i = 0;
    do {
       i++;
      document.write(i + "<br>");
    } while (i<10);
  </script>
</body>
</html>
```

break et continue

Vous pouvez utiliser les instructions `break` et `continue` pour interrompre l'exécution normale d'une boucle. Nous avons déjà rencontré l'instruction `break` dans le contexte d'une instruction `switch`, où elle permet de quitter la séquence à la suite d'une correspondance fructueuse.

Dans une boucle, `break` agit de la même manière. Autrement dit, elle quitte immédiatement la boucle, que les conditions de sortie de celle-ci soient ou non remplies.

Sur l'exemple du Listing 6.7, le jeu de recherche de mot fonctionne comme ci-dessus, mais cette fois la boucle se termine immédiatement si aucune valeur n'est entrée.

LISTING 6.7 : Utiliser break dans une boucle while.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
  <title>Trouvez le mot !</title>
</head>
<body>
  <script>
    var guessedWord = prompt("Quel est le mot auquel je pense ?");
    while (guessedWord != "sandwich") {
       if (guessedWord =="") {break;} // quitte tout de suite la boucle
si
                                          l'utilisateur n'a rien entré
      prompt("Non. Mauvaire réponse. Essayez encore.");
      }
      alert("Félicitations  ! C'est le bon mot !");
  </script>
</body>
</html>
```

L'instruction `continue` provoque l'arrêt de l'itération courante de la boucle et demande au programme de passer directement à la suivante. Les instructions qui suivent `continue` dans la boucle sont donc ignorées.

Le Listing 6.8 montre un programme qui compte de 1 à 20, mais en affichant seulement les nombres pairs. Remarquez qu'il utilise l'opérateur modulo pour tester la valeur courante afin de déterminer si elle est divisible par deux.

LISTING 6.8 : Utiliser l'instruction continue pour afficher une suite de nombres pairs.

```
<html>
<head>
  <title>Afficher uniquement les nombres pairs</title>
</head>
<body>
  <script>
    for (var i = 0; i <= 20; i++){
      if (i%2 != 0){
         continue;
      }
      document.write (i + " est un nombre pair.<br>");
    }
  </script>
</body>
</html>
```

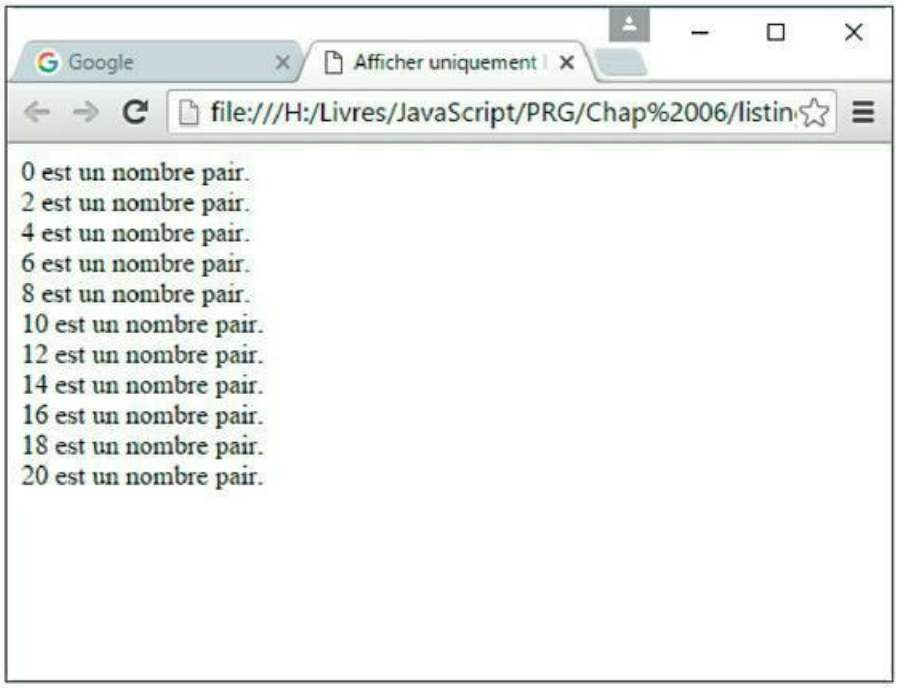

Google
Afficher uniquement
file:///H:/Livres/JavaScript/PRG/Chap%2006/listin
0 est un nombre pair.
2 est un nombre pair.
4 est un nombre pair.
6 est un nombre pair.
8 est un nombre pair.
10 est un nombre pair.
12 est un nombre pair.
14 est un nombre pair.
16 est un nombre pair.
18 est un nombre pair.
20 est un nombre pair.

FIGURE 6.6 : Un affichage de nombres pairs.

Lorsqu'elle est employée de cette manière, `continue` peut remplacer une clause `else`.

La Figure 6.6 illustre ce que produit le programme du Listing 6.8.

Les instructions `break` et `continue` peuvent être utiles, mais elles sont aussi dangereuses. Leur petite taille et leur grande puissance font qu'elles se dissimulent facilement au milieu du code, tout en gardant leurs forces. Pour cette raison, certains programmeurs considèrent que les utiliser à l'intérieur d'une boucle est une mauvaise pratique.

Organiser votre code JavaScript

DANS CETTE PARTIE...

Découvrir comment travailler avec les fonctions (pour en découvrir d'autres, voyez l'article concernant Underscore à l'adresse www.dummies.com/extras/codingwithjavascript)

Créer et utiliser des objets

Chapitre 7

Devenir fonctionnel

DANS CE CHAPITRE :

- **Écrire des fonctions**
- **Documenter les fonctions**
- **Passer des paramètres**
- **Retourner des valeurs**
- **Organiser les programmes avec des fonctions**

« Ma fonction, c'est d'écrire. Sans cela, je tomberai et je mourrai. C'est quelque chose d'organique, presque d'intestinal. Mais aussi très glamour. »

Charles Bukowski

Les fonctions vous aident à réduire la répétition du code en transformant des séquences utilisées fréquemment en composants réutilisables. Dans ce chapitre, vous écrirez plusieurs fonctions et vous vous en servirez pour réaliser facilement et avec le sourire des tâches qui seraient sinon fastidieuses.



N'oubliez pas de visiter le site Web compagnon de ce livre pour y télécharger les exemples de ce chapitre !

Comprendre la fonction des fonctions

Les *fonctions* sont comme des miniprogrammes à l'intérieur de vos programmes. Elles servent à prendre en charge des tâches particulières qui devraient sinon être recopiées manuellement plusieurs fois, voire de multiples fois, en différents endroits du code.

Si vous avez bien lu les chapitres précédents, vous avez déjà vu des fonctions en action. L'exemple qui suit est une fonction toute simple qui, quand elle est exécutée, ajoute la lettre z à la fin d'une chaîne de caractères.

```
function addZ(aString) {
  aString += "z";
  return aString;

}
```

Pour tester cette fonction, suivez ces étapes :

1. **Ouvrez la console JavaScript dans Chrome.**
2. **Saisissez la fonction.**

Vous pouvez effectuer un copier/coller, ou encore appuyer sur la combinaison Maj + Entrée après chaque ligne pour créer un saut de ligne sans exécuter le code.

3. **Appuyez sur Entrée une fois l'accolade finale saisie.**

   La console va répondre en affichant le mot `undefined`.

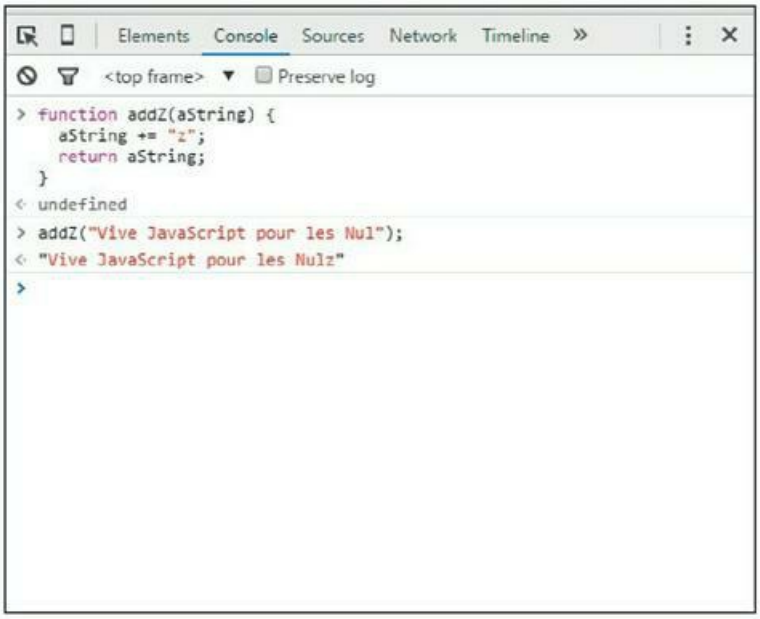


FIGURE 7.1 : Votre première fonction dans la console JavaScript de Chrome.

4. **Tapez maintenant la commande suivante, en la terminant par un appui sur la touche Entrée :**

   ```
   addZ("Vive JavaScript pour les Nul") ;
   ```

   Le résultat est illustré sur la Figure 7.1.

Les fonctions constituent une partie fondamentale de JavaScript. Et elles ont des tas de règles et de pouvoirs spéciaux auxquels vous devez faire particulièrement attention en tant que développeur JavaScript. Mais ne vous inquiétez pas s'y vous avez du mal à en mémoriser toutes les subtilités. Cela demande une certaine expérience, puisque vous devez comprendre un certain nombre de concepts plus ou moins abstraits. N'hésitez donc pas à relire plusieurs fois ce chapitre si vous pensez que c'est nécessaire. Et si tout devient clair pour vous, poursuivez votre voyage sans vous poser plus de questions !

Utiliser la terminologue des fonctions

Les programmeurs utilisent des tas de mots qu'il est important de comprendre lorsqu'il s'agit de fonctions. Nous allons employer ces mots de manière intensive dans ce chapitre et dans la suite de ce livre. Dans les sections qui suivent, vous allez donc découvrir ce langage que vous devrez vous-même parler lorsque vous allez travailler avec les fonctions.

Définir une fonction

Lorsqu'une fonction apparaît dans du code JavaScript, elle ne s'exécute pas. Elle est simplement créée, et est disponible pour répondre présent lorsqu'on a besoin d'y faire appel. Le fait de « poser » cette sorte de brique de construction s'appelle *définir* une fonction.



Vous n'avez besoin de définir une fonction dans un programme ou une page Web qu'une seule fois. Si vous le faisiez une seconde fois, cependant, JavaScript ne se plaindrait pas. Il utiliserait simplement la version dont la définition est la plus récente.

Voici un exemple de définition de fonction :

```
var maFonction = new Function() {
}  ;
```

Ou encore :

```
function maFonction(){
};
```

Définir l'en-tête de la fonction

Il s'agit de la partie de la définition de la fonction qui contient le mot-clé `function`, le nom de la fonction et des parenthèses. Par exemple :

```
function maFonction ()
```

Le corps de la fonction

Le *corps de la fonction* est constitué des instructions qui sont placées entre les accolades. Par exemple :

```
{
// corps de la fonction
}
```

Appeler une fonction

Pour utiliser une fonction, vous l'*appelez*. Cet appel provoque l'exécution des instructions qui se trouvent dans le corps de la fonction. Par exemple :

```
maFonction ;
```

Définir des paramètres et passer des arguments

Les *paramètres* sont les noms des morceaux de données qui sont fournis à une fonction lorsque vous l'appelez. Les *arguments* sont les valeurs que vous communiquez aux fonctions. Lorsqu'une fonction est appelée avec des arguments (d'une manière cohérente avec les paramètres spécifiés pour celle-ci), ceux-ci sont dits être *passés* à la fonction.

La syntaxe servant à définir un paramètre se présente ainsi :

```
function maFonction (paramètre) {
```

La syntaxe servant à passer un argument lors de l'appel à une fonction se présente ainsi :

```
maFonction (monArgument) ;
```

Retourner une valeur

En plus de pouvoir accepter des entrées provenant du monde extérieur, les fonctions peuvent aussi renvoyer des valeurs lorsque leur exécution se termine. Dans ce cas, on dit que la fonction *retourne* une valeur.

Pour qu'une fonction communique un résultat au monde extérieur, vous utilisez le mot-clé `return`. Par exemple :

```
return maValeur ;
```

Les fonctions, c'est tout bénéfice !

Le Listing 7.1 vous propose un programme qui additionne des nombres. Il s'exécute sans problème, et fait exactement ce pour quoi il est prévu en utilisant à cet effet une boucle `for…in` (voir le Chapitre 6).

LISTING 7.1 : Ajouter des nombres avec une boucle for…in.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
<title>Trouvez le total</title>
</head>
<body>
  <script>
    var myNumbers = [2,4,2,7];
    var total = 0;
    for (oneNumber in myNumbers){
    total = total + myNumbers[oneNumber];
    }
    document.write(total);
  </script>
</body>
</html>
```

Par contre, si nous avions de multiples séries de nombres à additionner, il nous faudrait écrire autant de boucles qu'il y a d'opérations à effectuer.

Le Listing 7.2 reprend cet exercice, mais il transforme la boucle du Listing 7.1 en fonction, de manière à pouvoir trouver la somme des éléments de plusieurs tableaux.

LISTING 7.2 : Une fonction pour additionner les valeurs contenues dans un tableau.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
<title>Trouvez la somme</title>
</head>
<body>
  <script>
    /**
     * Ajoute les éléments dans un tableau et retourne le total
     *@param {Array.<number>} numbersToAdd
     *@return {Number} sum
     */
     function addNumbers(numbersToAdd) {
     var sum = 0;
     for (oneNumber in numbersToAdd) {
      sum = sum + numbersToAdd[oneNumber];
     }
     return sum;
     }

     var myNumbers = [2,4,2,7];
     var myNumbers2 = [3333,222,111];
     var myNumbers3 = [777,555,777,555];
     var sum1 = addNumbers(myNumbers);
     var sum2 = addNumbers(myNumbers2);
     var sum3 = addNumbers(myNumbers3);

     document.write(sum1 + "<br>");
     document.write(sum2 + "<br>");
     document.write(sum3 + "<br>");

  </script>
</body>
</html>
```



Le bloc de commentaire dans le Listing 7.2 suit le format spécifié par le système de documentation JavaScript, JSDoc. Cela rend non seulement vos programmes plus faciles à lire, mais permet de plus de générer automatiquement des fichiers de documentation. Voyez à ce sujet l'encadré « Documenter JavaScript avec JSDoc », ainsi que le site Web http://usejsdoc.org.

Les fonctions permettent de gagner beaucoup de temps, d'effort et de place. Écrire une fonction utile peut au départ prendre plus de temps que du code JavaScript « classique ». Mais, à plus long terme, vos programmes seront mieux organisés, plus clairs, et vous vous épargnerez de grandes douleurs morales en prenant l'habitude d'écrire des fonctions.

DOCUMENTER JAVASCRIPT AVEC JSDOC

Une bonne pratique consiste à toujours documenter votre code JavaScript en utilisant un système standardisé. Le système le plus utilisé, et qui est donc, *de facto*, un standard, s'appelle JSDoc.

JSDoc est un langage de balises simple qui peut être inséré dans les fichiers JavaScript. JSDoc en est actuellement à sa troisième version, et il est basé sur le système JavaDoc qui sert à documenter le code écrit dans le langage de programmation Java.

Une fois que vous avez annoté votre code JavaScript avec JSDoc, vous pouvez utiliser un générateur, par exemple jsdoc-toolkit, pour produire des fichiers de documentation HTML.

JSDoc place ses balises dans des blocs de commentaires spéciaux. En fait, la seule différence avec le code JavaScript standard est que ces commentaires commencent par /** et se terminent par */ (JavaScript se contente, lui d'un seul astérisque au début du commentaire). Cela vous permet donc d'insérer des notes ordinaires, sans que celles-ci soient reprises dans la documentation générée.

Les balises spéciales de JSDoc se reconnaissent en particulier à la présence du caractère @ devant un mot-clé. Les balises les plus courantes sont listées ci-après :

@author	Nom du développeur
@constructor	Indique qu'une fonction est un constructeur
@deprecated	Indique que la méthode est périmée
@exception	Décrit une exception lancée par une méthode. synonyme de @ throws
@exports	Spécifie un membre qui est exporté par le module
@param	Décrit un paramètre de la méthode
@private	Indique qu'un membre est privé
@return	Décrit une valeur retournée. Synonyme de @returns
@returns	Voir @return
@see	Enregistre une association avec un autre objet
@this	Spécifie les types de l'objet auxquels le mot-clé this fait référence à l'intérieur d'une fonction
@throws	Voir @throw
@version	Indique le numéro de version d'une bibliothèque.

Écrire des fonctions

Une fonction doit être déclarée en suivant un ordre précis, de cette manière :

» Le mot-clé Function.

- Le nom de la fonction.
- Des parenthèses qui peuvent contenir un ou plusieurs paramètres.
- Une paire d'accolades contenant les instructions que la fonction doit exécuter.

Parfois, le seul rôle d'une fonction consiste à envoyer un message à l'écran dans une page Web. Un exemple typique est l'affichage de la date courante, comme ceci :

```
function laDate(){
  var maintenant = new Date();
  document.write(maintenant.toDateString());
}
```

Pour tester cette fonction, suivez ces étapes :

1. **Ouvrez la console JavaScript de Chrome.**
2. **Tapez la fonction dans la console.**

   Servez-vous de la combinaison Maj + Entrée à la fin de chaque ligne pour ne pas exécuter le code correspondant de manière anticipée.
3. **Appuyez sur Entrée à la fin de la dernière ligne.**

   Rien ne se passe à ce stade, si ce n'est que la console affiche le mot `undefined`. Cela signifie que la fonction a été acceptée, mais qu'elle n'a retourné aucune valeur.
4. **Appelez la fonction en tapant son nom, suivi d'une paire de parenthèses et d'un point-virgule, comme ceci :**

   ```
   laDate () ;
   ```

   La fonction affiche la date et l'heure courantes (au format anglo-saxon), et la console affiche le terme `undefined`, puisque la fonction n'a pas de valeur de retour.



Puisque la valeur renvoyée par la console est `undefined`, ce mot-clé est techniquement parlant une valeur de retour.

Retourner des valeurs

Dans l'exemple de la section précédente, nous avons créé une fonction qui se contente d'afficher un message dans la fenêtre du navigateur Web. Une fois l'instruction `.write` exécutée, il n'y a plus rien de spécial à faire, et la fonction se termine à ce stade. Le programme principal, de son côté, reprend son cours normal au niveau de l'instruction qui suit l'appel à la fonction.

La plupart des fonctions sont conçues pour retourner une valeur (autre que `undefined`) une fois leur travail terminé. Cette valeur peut alors être utilisée dans la suite du programme.

Le Listing 7.3 montre une fonction retournant une valeur. Celle-ci est alors affectée à une variable, puis affichée dans la console.

LISTING 7.3 : Exemple de fonction retournant une valeur.

```
function getHello(){
  return "Hello  !";
}

var helloText = getHello();
console.log (helloText);
```

Dans une fonction, l'instruction `return` est généralement la dernière. Lorsqu'elle est rencontrée, l'exécution de la fonction se termine. Vous pouvez utiliser l'instruction `return` aussi bien pour retourner un littéral (comme "Hello !" ou 3) que la valeur d'une variable ou d'une expression, un tableau ou un objet, et même une autre fonction. Par exemple, le Listing 7.4 propose une fonction chargée de calculer le périmètre d'un cercle de rayon prédéfini.

LISTING 7.4 : Retourner la valeur produite par une expression.

```
function getCircumference(){
  var radius = 12;
  return 2 * Math.PI * radius;
}
```

Bien entendu, il serait plus intéressant de passer ce rayon en argument, comme le propose le Listing 7.4a.

LISTING 7.4A : Retourner la valeur produite par une expression (variante).

```
function getCircumference(radius){
  return 2 * Math.PI * radius;
}

console.log (getCircumference(12));
```

Passer et utiliser des arguments

Pour que les fonctions soient capables d'effectuer un certain travail avec différentes entrées, il faut que le programmeur dispose d'un moyen de spécifier celles-ci. Nous en avons d'ailleurs déjà rencontré un exemple (reportez-vous au Listing 7.2). Pour cela, les parenthèses qui suivent le nom de la fonction dans la définition de celle-ci peuvent contenir un ou plusieurs paramètres.



La différence entre *paramètres* et *arguments* peut sembler un peu confuse au départ. Voici comment les choses fonctionnent :

» Les *paramètres* sont les noms que vous spécifiez dans la définition de la fonction.

» Les *arguments* sont les valeurs que vous passez à cette fonction. Ils prennent alors le nom des paramètres correspondants.

Lorsque vous appelez une fonction, vous incluez des données (les arguments) là où la définition de la fonction contient des paramètres.



L'ordre dans lequel les arguments sont passés doit être conforme à celui des paramètres dans la définition de la fonction.

Dans la fonction qui suit, nous définissons deux paramètres pour une fonction appelée myTacos (NdT : Les Américains sont très friands de plats mexicains…).

```
function myTacos(meat,produce){
...
}
```

Lorsque vous appelez cette fonction, vous devez inclure vos arguments dans le même ordre, et avec le même type, que les paramètres qui ont été définis, comme ceci :

```
myTacos ("boeuf", "oignons") ;
```

Ces valeurs deviennent celles de variables locales à la fonction, donc des paramètres. C'est le nom de ceux-ci qui sont ensuite utilisés dans le corps de la fonction.

Le Listing 7.5 étend la fonction `myTacos` pour afficher dans la console la valeur des deux arguments. Passer un argument est comme déclarer une variable avec `var` à l'intérieur de la fonction, si ce n'est que ces valeurs peuvent provenir d'une source extérieure.

LISTING 7.5 : Faire référence aux arguments dans une fonction en utilisant les noms des paramètres.

```
function myTacos(meat,produce){
  console.log(meat); // écrit "boeuf"
  console.log(produce); // écrit "oignons"
}

myTacos("boeuf","oignons");
```



Vous pouvez spécifier jusqu'à 255 paramètres dans la définition d'une fonction. Mais, bien entendu, atteindre ou même seulement s'approcher de cette limite serait très inhabituel ! Rien que pour des raisons de clarté et de compréhension, il vaudrait mieux rechercher une autre méthode si vous vous trouvez un jour dans une telle situation.

Passer des arguments par valeur

Si vous utilisez une variable avec l'un des types de données primitifs de JavaScript pour passer votre argument, on dit que celui-ci est passé *par valeur*. Cela signifie que la nouvelle variable créée à l'intérieur de la fonction est totalement distincte de celle qui fournit l'argument. En d'autres termes, quoi qu'il se passe ensuite dans le corps de la fonction, la variable extérieure ne sera pas modifiée.



Les types de données primitifs en JavaScript sont les chaînes (`string`), les nombres (`number`), les booléens (`Boolean`), `undefined` et `null`.

Sur le Listing 7.6, vous pouvez voir plusieurs variables créées, remplies d'une certaine valeur, puis passées à une fonction. Dans ce cas, les paramètres de la fonction possèdent le même nom que celui utilisé pour définir les arguments. C'est bien sûr tout à fait volontaire, pour vous montrer que le contenu des arguments ne change pas lorsque la fonction est exécutée.

LISTING 7.6 : Passage d'arguments par valeur à une fonction.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
<title>Arguments Passés Par Valeur</title>
</head>
<body>
  <script>
    /**
    * Incrémente deux nombres
    * @param {number} number1
    * @param {number} number2
    */
   function addToMyNumbers(number1,number2){
     number1++;
     number2++;
     console.log("nombre 1: " + number1);
     console.log("nombre 2: " + number2);
    }

    var number1 = 3;
    var number2 = 12;

    addToMyNumbers(number1,number2); // passe les arguments

    console.log("nombre original 1 : " + number1);
    console.log("nombre original 2 : " + number2);
  </script>
</body>
</html>
```

La Figure 7.2 illustre la sortie produite dans la console JavaScript de Chrome.

Passer des arguments par référence

Alors que les types primitifs JavaScript contenus dans des variables sont passés par valeur, les objets sont, eux, passés *par référence*. Cela signifie que, si vous passez un objet en tant qu'argument à une fonction, toute modification apportée à cet objet à l'intérieur de la fonction le changera aussi à l'extérieur de celle-ci. Les implications de ce comportement sont traitées dans le Chapitre 8.

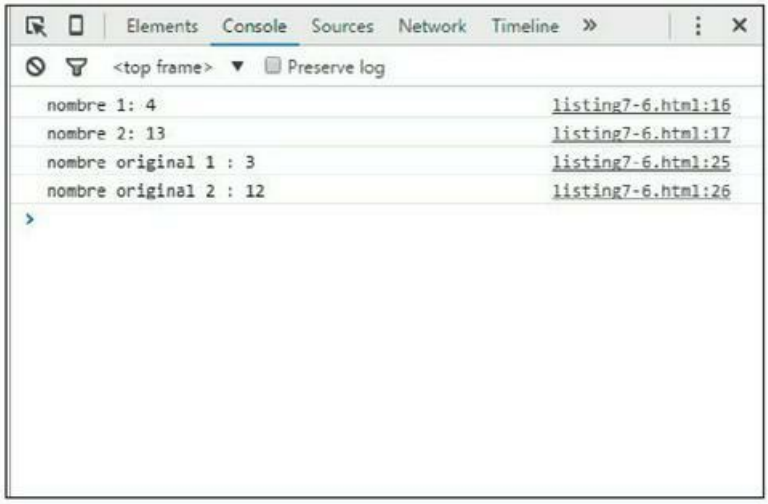


FIGURE 7.2 : Les variables extérieures à une fonction ne sont pas affectées par ce qui se passe à l'intérieur de celle-ci.

Appeler une fonction sans passer tous les arguments

Il n'est pas nécessaire d'appeler une fonction avec la totalité des paramètres déclarés dans sa définition. Si, par exemple, cette définition contient trois paramètres, et que vous l'appelez en ne spécifiant que deux arguments, le troisième créera simplement une variable possédant comme valeur `undefined`.

Définir la valeur par défaut des paramètres

Si vous voulez que la valeur d'arguments non spécifiés devienne autre chose que `undefined`, vous pouvez définir des valeurs par défaut. La méthode la plus courante dans ce cas consiste à tester les paramètres transmis à l'intérieur de la fonction, afin de pouvoir définir une valeur par défaut si le type de l'argument est `undefined`.

Sur le Listing 7.7, par exemple, la fonction possède un paramètre. Un test interne recherche si l'argument est `undefined`. Dans ce cas, on lui affecte une valeur par défaut.

LISTING 7.7 : Définir des valeurs par défaut pour les arguments.

```
function welcome(yourName){
  if (typeof yourName === 'undefined'){
  yourName = "ami";
}
```

Dans la prochaine version de JavaScript, appelée ECMAScript 6, il sera possible de définir ces valeurs par défaut directement dans l'en-tête des fonctions, comme l'illustre le Listing 7.8.

LISTING 7.8 : Définir des arguments par défaut dans l'en-tête d'une fonction.

```
function welcome(yourName = "ami") {
  document.write("Bonjour, " + youName);
}
```



À l'heure actuelle, ECMAScript 6 n'est pas encore pris en charge par tous les navigateurs. Il est donc possible que cette méthode échoue avec certains utilisateurs de vos programmes. Pour cette raison, il est préférable de continuer à utiliser pour l'instant la méthode employée dans le Listing 7.7.

Appeler une fonction avec plus d'arguments que de paramètres

Si vous passez plus d'arguments que de paramètres, les variables locales en trop ne seront pas créées, puisque la fonction n'a aucun moyen de savoir de quoi il s'agit.

Il existe cependant une astuce permettant de retrouver la valeur des arguments en trop : l'objet `arguments`.

Plonger dans la liste des arguments

Si vous ne savez pas combien d'arguments seront passés à une fonction, vous pouvez faire appel à l'objet `arguments`, qui est automatiquement associé aux fonctions par JavaScript, pour retrouver tous les arguments qui ont été passés et vous en servir pour effectuer les traitements nécessaires.

L'objet `arguments` contient un tableau où se trouvent tous les arguments passés à une fonction. En parcourant ce tableau dans une boucle de type `for` ou `for…infor… in`, vous pouvez utiliser chaque argument, même si le nombre de ces arguments change chaque fois que la fonction est appelée.

Le Listing 7.9 illustre l'emploi de l'objet `arguments` pour présenter un message de bienvenue adapté aux personnes qui ont un unique prénom, comme à celles qui ont un prénom composé .

LISTING 7.9 : Utiliser l'objet arguments pour personnaliser un message.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
<title>Message de Bienvenue</title>
</head>
<body>
  <script>

    /**
    *Message de bienvenue ajustable
    */
    function flexibleWelcome(){
    var welcome = "Bienvenue, ";
    for (i = 0; i < arguments.length; i++) {
     welcome = welcome + arguments[i] + "";
    }
   return welcome;
    }

    document.write(flexibleWelcome("Christopher-","James-","Phoenix-
","Minnick") +
      "<br>");
    document.write(flexibleWelcome("Eva-","Ann-","Holland") + "<br>");

  </script>
</body>
</html>
```

Portée des fonctions

Les variables créées dans une fonction en lui passant des arguments, ou en utilisant le mot-clé `var`, ne sont disponibles que dans cette fonction. On appelle cela la *portée* d'une fonction. Ces variables sont détruites lorsque la fonction se termine.

Cependant, si vous créez une variable dans une fonction sans spécifier le mot-clé `var`, elle devient *globale*, et il est possible d'y accéder et de la modifier partout dans votre programme.



Créer accidentellement une variable globale dans une fonction est la source d'un grand nombre d'erreurs en JavaScript. Il est donc plus que vivement recommandé de toujours définir avec précision la portée des variables, et en particulier de ne créer des variables globales que quand c'est absolument nécessaire.

Le cas de la fonction anonyme

La partie *nom* dans l'en-tête d'une définition de fonction n'est pas obligatoire. Autrement dit, vous pouvez créer des fonctions sans nom. Cela peut paraître très bizarre. Comment faire pour appeler une fonction qui n'a pas de

nom ?

Cependant, des fonctions anonymes peuvent être affectées à des variables lorsque celles-ci sont créées, ce qui vous offre finalement les mêmes possibilités qu'avec les fonctions nommées. Par exemple :

```
var choseAFaire = function(uneChose) {
  document.write("Je vais faire cette chose : " + uneChose);
}
```

Différences entre fonctions anonymes et fonctions nommées

Il existe quelques différences importantes, et parfois utiles, entre créer une fonction nommée et affecter une fonction anonyme à une variable. La première est qu'une fonction anonyme affectée à une variable n'existe et ne peut être appelée qu'après l'exécution de cette affectation par le programme. Les fonctions nommées, par contre, peuvent être appelées n'importe où dans le programme.

La seconde différence est que vous pouvez changer la valeur d'une variable et lui affecter une autre fonction à n'importe quel moment. Cela donne aux fonctions anonymes une souplesse particulière.

Fonctions anonymes auto-exécutables

Une autre application de ce type de fonction est ce que l'on appelle les fonctions *auto-exécutables*. On appelle ainsi une fonction anonyme qui s'exécute dès qu'elle est créée.

Pour transformer une fonction anonyme en fonction toujours aussi anonyme, mais auto-exécutable, il vous suffit de la placer dans des parenthèses, puis d'ajouter une paire de parenthèses et un point-virgule.

Le bénéfice à attendre de cette technique est que les variables définies à l'intérieur d'une fonction anonyme auto-exécutable sont détruites lorsque la fonction se termine. De cette manière, vous pouvez éviter des conflits entre des noms de variables, et vous échappez aussi à l'occupation de la mémoire de l'ordinateur par ces variables une fois que vous n'en avez plus besoin. Le Listing 7.10 vous propose un exemple d'utilisation d'une fonction anonyme auto-exécutable.

LISTING 7.10 : Utiliser une fonction anonyme auto-exécutable.

```
var myVariable = "Je me trouve en dehors de la fonction.";
  (function() {
  var myVariable = "Je me trouve dans cette fonction anonyme";
  document.write(myVariable + "<br>");
  })();
document.write(myVariable);
```

Les programmeurs d'applications pour le Web utilisent régulièrement des fonctions anonymes pour produire de multiples effets modernes dans les pages Web. Vous en apprendrez plus sur tout cela dans les Chapitres 15 et 16.

On recommence depuis le début (la récursivité)

Vous pouvez appeler des fonctions depuis du code qui leur est extérieur, mais aussi à l'intérieur d'autres fonctions. Il est même possible d'appeler une fonction à l'intérieur d'elle-même. C'est ce que l'on appelle la *récursivité*

(certains programmeurs préfèrent *récursion*).

La récursivité peut s'employer dans la plupart des cas à la place d'une boucle, si ce n'est que les instructions sont répétées à l'intérieur d'une fonction.

Le Listing 7.11 montre un exemple simple de récursivité. Cependant, cette fonction a un gros problème. Pouvez-vous le trouver ?

LISTING 7.11 : Une fonction récursive fatalement désastreuse.

```
function squareItUp(startingNumber) {
  square = startingNumber * startingNumber;
  console.log(square);
  squareItUp(square);
}
```

Est-ce que vous voyez la fin de l'histoire ? Non, nulle part. Cette fonction boucle indéfiniment. Elle va simplement empiler des hordes de nombres sans jamais s'arrêter, et ce jusqu'à ce que vous arriviez à refermer la fenêtre du navigateur.



En fait, en l'état, cette fonction va probablement provoquer un plantage de votre navigateur, si ce n'est de votre ordinateur. Elle ne va provoquer aucun dommage irréparable, bien entendu, mais elle est suffisamment dangereuse pour que vous vous contentiez de lire son code sans essayer de l'exécuter.

Le Listing 7.12 améliore cette fonction en fournissant une condition servant à indiquer à la fonction qu'elle a fini son travail et qu'elle doit s'arrêter. Toute fonction récursive doit comporter une telle condition (ou *condition de base*), quelle qu'elle soit.

LISTING 7.12 : Une fonction récursive calculant des carrés jusqu'à ce que le résultat dépasse un million.

```
function squareItUp(startingNumber) {
  square = startingNumber * startingNumber;
  if (square > 1000000) {
    return square;
  } else {
  return squareItUp(square);
}
```

C'est déjà mieux. Mais cette fonction a toujours un gros problème. Que se passerait-il si quelqu'un lui passait une valeur négative, nulle ou bien inférieure ou égale à 1 (ou autre chose qu'un nombre) ? Le résultat serait une boucle infinie, puisque la condition ne serait jamais vérifiée. La condition de terminaison, ou d'arrêt, doit pouvoir traiter toutes les situations possibles afin de provoquer la sortie immédiate de la fonction dans le cas où cette condition ne peut pas être vérifiée. C'est ce que fait la variante proposée dans le Listing 7.13. Elle débute cette fois par l'examen d'une condition d'arrêt qui stoppera la fonction dans le cas où elle n'est pas vérifiée.

LISTING 7.13 : Une fonction récursive avec conditions de base et de terminaison.

```
function squareItUp(startingNumber) {

  // Conditions d'arrêt, saisie invalide
  if ((typeof startingNumber != ‘number') || (startingNumber <= 1)) {
  return - 1
  }

  square = staringNumber * startingNumber;

  //Condition de base
  if (square > 1000000) {
    return square; // Retourne la valeur finale
    } else { // Si la condition de base n'est pas remplie, recommencer.
    return squareItUp(square);
  }

}
```

Des fonctions dans des fonctions

Il est possible de déclarer des fonctions à l’intérieur d’autres fonctions. Le Listing 7.14 illustre cette technique, et montre comment elle peut affecter la portée des variables créées à l’intérieur des fonctions.

LISTING 7.14 : Déclarer des fonctions à l’intérieur de fonctions.

```
function turnIntoAMartian(myName) {

  function recallName(myName) {
    var martianName = myName + " Martien";
  }
  recallName(myName);
  console.log(martianName); // Renvoie une valeur indéfinie
}
```

Cet exemple démontre en quoi une déclaration de fonction à l’intérieur d’une autre modifie la portée des variables. Les variables déclarées dans la fonction intérieure ne sont pas directement accessibles à la fonction de rang supérieur. Pour obtenir leur valeur, il faut une instruction de retour, comme l’illustre le Listing 7.15.

LISTING 7.15 : Valeurs de retour produites par une fonction intérieure.

```
function turnIntoAMartian(myName) {

  function recallName(myName) {
    var martianName = myName + " Martien";
    return martianName;
  }
  var martianName = recallName(myName);
  console.log(martianName);
}
```

Chapitre 8

Créer et utiliser des objets

DANS CE CHAPITRE :

- **Comprendre les objets**
- **Utiliser des propriétés et des méthodes**
- **Créer des objets**
- **Comprendre la notation point (.)**
- **Travailler avec les objets**

« Nous ne pouvons rien faire avec un objet qui n'a pas de nom. »

Maurice Blanchot

Dans ce chapitre, nous allons vous expliquer pourquoi vous devriez utiliser des objets, comment les utiliser, et quels pouvoirs spéciaux ils possèdent pour améliorer vos programmes et votre travail de programmeur.



N'oubliez pas de visiter le site Web compagnon de ce livre pour y télécharger les exemples de ce chapitre !

L'obscur objet de mon désir…

En plus de ses cinq types de données primitifs (décrits dans le Chapitre 3), JavaScript possède également un type particulier appelé *objet*. Les objets de JavaScript servent à *encapsuler* des données et des fonctionnalités qui font de ces objets des composants réutilisables.

Pour comprendre ce que sont les objets et comment ils travaillent, il est intéressant de faire un parallèle entre objets JavaScript et objets du monde réel. Prenons par exemple une guitare.

Une guitare se définit par des matériaux qui font qu'elle est ce qu'elle est, et il y a aussi des choses qu'elle est capable de faire. Voici quelques faits concernant la guitare que nous allons utiliser pour cet exemple :

- Elle a six cordes.
- Elle est noire et blanche.
- Elle est électrique.
- Son corps est compact

Parmi ce que la guitare est capable de faire (ou plutôt de ce qu'on peut lui faire faire), il y a ceci :

- Gratter les cordes
- Augmenter le volume

- Baisser le volume
- Retendre les cordes
- Ajuster la tonalité
- Changer les cordes

Si cette guitare était un objet JavaScript plutôt qu'une véritable guitare du monde réel, les choses qu'elle fait s'appelleraient des *méthodes*, et ses caractéristiques physiques seraient ses *propriétés*.

Les méthodes et les propriétés des objets s'écrivent de la même manière, c'est-à-dire sous la forme d'un couple nom/valeur, les deux étant séparés par le caractère : . Lorsqu'une propriété est définie par une fonction, on dit qu'il s'agit d'une méthode.



En réalité, tout ce qui définit un objet est une propriété. La notion de méthode sert à distinguer les propriétés qui sont définies à l'aide d'une fonction.

Le Listing 8.1 illustre la manière dont les propriétés de l'objet Guitare pourraient s'écrire en JavaScript.

LISTING 8.1 : Un objet Guitare en JavaScript.

```
var guitare = {
 couleurCorps: "noir",
 couleurPlateau: "blanc",
 nombreDeCordes: 6,
 marque: "Yamaha",
 typeCorps: "compact",
 gratter: function() { …},
 tonalité: function() { … }
};
```

Créer des objets

JavaScript propose deux procédés pour créer des objets :

- La méthode littérale d'écriture d'objet
- La méthode de création par constructeur

Le choix à faire dépend des circonstances. Dans les sections qui suivent, vous allez découvrir les pour et les contre de chaque méthode, et quand il vaut mieux choisir l'une plutôt que l'autre.

Définir des objets : la méthode littérale

Cette technique de création d'objet débute par une définition standard de variable, donc en commençant par le mot-clé `var` suivi du nom de l'objet et de l'opérateur d'affectation :

```
var personne =
```

La suite est différente. Vous placez entre des accolades des couples nom/valeur séparés d'une virgule, comme ceci :

```
var personne = {yeux : 2, pieds : 2, mains : 2, couleurYeux : "bleu"} ;
```

Si vous ne connaissez pas les propriétés de l'objet au moment de sa création, ou si votre programme nécessite l'ajout ultérieur de nouvelles propriétés, vous pouvez créer cet objet avec un nombre minimum de propriétés, voire aucune. Il vous suffit d'en compléter plus tard la liste :

```
var personne = {};
personne.yeux = 2;
personne.cheveux = "blonds";
```

Les méthodes des exemples utilisés dans les chapitres précédents ont surtout été utilisées pour afficher du texte. Nous avons en effet déjà eu à rencontrer de multiples fois les instructions `document.write` et `console.log`, ce qui fait que cette structure séparant le nom de l'objet et une de ses propriétés ou méthodes par un point devrait maintenant vous être familière. Cette notation est traitée plus en détail dans la section « Retrouver et définir les propriétés des objets », plus loin dans ce chapitre.

Un autre point à noter concernant les objets est que, comme les tableaux, leurs propriétés peuvent contenir différents types de données.



En fait, pour JavaScript, les tableaux et les fonctions sont des types d'objets, et les types de données que sont les nombres, les chaînes de caractères et les valeurs booléennes peuvent aussi être utilisés comme des objets. En d'autres termes, ils ont toutes les propriétés des objets et on peut leur assigner des propriétés de la même manière que pour les objets.

Définir des objets : la méthode par constructeur

La seconde technique consiste à définir un objet en utilisant le constructeur `Object`. Dans ce cas, la définition débute par `new Object`, suivi des propriétés de l'objet. Le Listing 8.2 illustre cette méthode.

LISTING 8.2 : Utiliser un constructeur d'objet.

```
var personne = new Object();
personne.pieds = 2;
personne.nom = "Julie";
personne.cheveux = "noir";
```

Cependant, cette façon de procéder n'est généralement pas considérée comme étant la meilleure. Il y a à cela plusieurs raisons :

- Elle implique davantage de saisie que la méthode littérale.
- Elle est moins efficace dans les navigateurs Web.
- Elle est plus difficile à lire que la méthode littérale.

Retrouver et définir les propriétés des objets

Une fois que vous avez créé un objet et défini ses propriétés, vous allez évidemment vouloir retrouver et modifier celles-ci. Vous disposez pour cela de deux variantes : la notation point et l'emploi de crochets droits.

La notation point

Dans cette notation, le nom de l'objet est suivi d'un point, puis du nom de la propriété que vous voulez lire ou définir.

Pour créer une nouvelle propriété de l'objet `personne` appelée `nomFamille`, par exemple, ou encore pour modifier la valeur d'une propriété `nomFamille` existante, vous utilisez dans ce cas une syntaxe de ce genre :

```
personne.nomFamille = "Durand" ;
```

Si cette propriété n'existe pas encore, cette instruction va la créer. Si elle existe déjà, elle va modifier sa valeur.

Pour retrouver une certaine propriété avec cette syntaxe, vous écrivez en fait la même chose, mais à un emplacement différent dans l'instruction (on parle alors d'*accesseur*). Par exemple, si vous voulez concaténer les valeurs contenues dans les propriétés `nomFamille` et `prenom` d'un objet `personne`, et affecter le résultat à une nouvelle variable appelée `nomComplet`, vous allez écrire ceci :

```
var nomComplet = personne.prenom + " " + personne.nomFamille
```

Il est également possible, toujours par exemple, d'utiliser simplement un accesseur exactement comme s'il s'agissait d'une variable quelconque, comme dans :

```
document.write (person.nomFamille) ;
```



La notation point est généralement plus rapide à taper, et plus facile à lire, pour retrouver ou définir les propriétés des objets.

La notation avec crochets droits

Cette notation repose sur le même principe que la précédente, si ce n'est que le point est remplacé par des crochets droits, le nom de la propriété étant spécifié entre guillemets, comme ceci :

```
personne["nomFamille"] = "Durand" ;
```

Cette technique offre davantage de possibilités que la notation point. La principale différence est qu'elle vous permet d'utiliser des noms de variables entre les crochets droits pour le cas où le nom de la propriété ne serait pas connu lorsque vous écrivez votre programme.

L'exemple qui suit fait exactement la même chose que le précédent, mais cette fois le nom littéral de la propriété est remplacé par celui d'une variable. De cette manière, vous pouvez créer une instruction qui peut servir dans de multiples circonstances, par exemple dans une boucle ou dans une fonction :

```
var personneProp = "nomFamille";
personne[personneProp] = "Durand";
```

Le Listing 8.3 montre un programme simple qui crée un objet appelé `myChair` dont la nature est de représenter une chaise. Les propriétés de cet objet sont parcourues dans une boucle, en demandant à chaque fois à l'utilisateur de saisir une valeur. Ensuite, un appel à la fonction `writeChairReceipt` affiche chacune des propriétés de l'objet ainsi que la valeur définie par l'utilisateur.

LISTING 8.3 : Un script de configuration de chaise.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
<title>Le Configurateur de Chaise</title>

</head>
<body>
  <script>
    var myChair = {"TissuAssise":"", "NombrePieds":"", "HauteurPied":""};

    function configureChair() {
    var userValue;
      for (var property in myChair) {
        if (myChair.hasOwnProperty(property)) {
          userValue = prompt("Entrez une valeur pour " + property);
          myChair[property] = userValue;
        }
      }
    }

    function writeChairReceipt() {
    document.write("<h2>Votre chaise aura la configuration suivante :
</h2>");
      for (var property in myChair) {
        if (myChair.hasOwnProperty(property)) {
          document.write(property + ": " + myChair[property] + "<br>");
        }
      }
    }
   configureChair();
    writeChairReceipt();
  </script>
</body>
</html>
```

Supprimer des propriétés

Vous pouvez supprimer des propriétés d'un objet en utilisant l'opérateur `delete`. Le Listing 8.4 illustre le fonctionnement de cet opérateur.

LISTING 8.4 : Utiliser l'opérateur delete.

```
var myObject = new Object(); // crée un nouvel objet

// ajoute des propriétés à l'objet
myObject.var1 = "la valeur";
myObject.var2 = "une autre valeur";
myObject.var3 = "encore une autre";

// supprime var2 de myObject
delete myObject.var2;

// essaie d'écrire la valeur de var2
document.write(var2 in myObject); // le résultat est une erreur
```

Travailler avec les méthodes

Les *méthodes* sont des propriétés dont la valeur est définie par une fonction. Pour définir une méthode, vous procédez exactement comme pour n'importe quelle autre fonction. La seule différence, c'est qu'une méthode est affectée à une propriété d'un objet. Le Listing 8.5 illustre la création d'un objet possédant plusieurs propriétés, dont l'une est une méthode.

LISTING 8.5 : Créer une méthode.

```
var sandwich = {
meat:"",
cheese:"",
bread:"",
condiment:"",
makeSandwich: function (meat,cheese,bread,condiment) {
  sandwich.meat = meat;
  sandwich.cheese = cheese;
  sandwich.bread = bread;
  sandwich.condiment = condiment;
  var mySandwich = sandwich.bread + ", " + sandwich.meat + ", " +
sandwich.cheese +
        ", " + sandwich.condiment;
  return mySandwich;
}
}
```

Pour appeler la méthode `makeSandwich` de l'objet `sandwich`, vous pouvez utiliser la notation point, exactement comme si vous accédiez à une propriété, mais en spécifiant des parenthèses après le nom de la méthode, comme l'illustre le Listing 8.6.

LISTING 8.6 : Appeler une méthode.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
<title>Préparez-moi un sandwich</title>
</head>
<body>
  <script>

    var sandwich = {
    meat:"",
    cheese:"",
    bread:"",
    condiment:"",
    makeSandwich: function (meat,cheese,bread,condiment) {
      sandwich.meat = meat;
      sandwich.cheese = cheese;
      sandwich.bread = bread;
      sandwich.condiment = condiment;
      var mySandwich = sandwich.bread + ", " + sandwich.meat + ", " +
sandwich.
        cheese + ", " + sandwich.condiment;
      return mySandwich;
      }
    }

    var sandwichOrder = sandwich.makeSandwich("jambon","cheddar","pain au
        cumin","moutarde forte");
    document.write (sandwichOrder);

  </script>
</body>
</html>
```

Utiliser this

Le mot-clé `this` est un raccourci qui permet de faire référence à l'objet concerné par une méthode. Sur le Listing 8.7, par exemple, chaque instance du nom de l'objet `sandwich` a été remplacée par `this`. Lorsque la fonction `makeSandwich` est appelée en tant que méthode de l'objet `sandwich`, JavaScript comprend que ce `this` fait référence à cet objet, et à aucun autre.

LISTING 8.7 : Utiliser this dans une méthode.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
<title>Préparez-moi un sandwich</title>
</head>
<body>
  <script>

  var sandwich = {
  meat:"",
  cheese:"",
  bread:"",
  condiment:"",
  makeSandwich: function (meat,cheese,bread,condiment) {
    this.meat = meat;
    this.cheese = cheese;
    this.bread = bread;
    this.condiment = condiment;
    var mySandwich = this.bread + ", " + this.meat + ", " + this.cheese +
", " +
        this.condiment;
    return mySandwich;
    }
  }

  var sandwichOrder = sandwich.makeSandwich("jambon","cheddar","pain au
      cumin","moutarde forte");
  document.write (sandwichOrder);

  </script>
</body>
</html>
```

Dans ce cas, le résultat produit par l'emploi du mot-clé `this` est exactement le même que dans l'exemple précédent.

Là où `this` devient très utile, c'est lorsque vous avez une fonction susceptible de s'appliquer à plusieurs objets. Ici, le mot-clé `this` fait référence à celui à partir duquel il a été appelé, plutôt que d'être associé à un objet spécifique.

Dans les sections suivantes, vous en apprendrez plus sur les constructeurs et l'héritage, deux fonctionnalités qui sont permises par cette humble instruction qu'est `this`.

Une méthode orientée objet pour devenir riche :

l'héritage

Lorsque vous créez des objets, vous n'êtes pas limité à quelque chose de tout à fait spécifique, comme votre guitare, votre voiture, votre chat ou votre sandwich. La véritable beauté des objets, c'est que vous pouvez les utiliser pour créer des types d'objets, ou si vous préférez des objets génériques, à partir desquels il est possible de produire des objets plus spécifiques.

Si vous avez lu les sections précédentes (et vous les avez lues), vous avez compris que la méthode de création `Object` avait ses avantages et ses inconvénients. Bien.

Reprenons cette technique de *constructeur* pour préciser cela :

```
var personne = new Object () ;
```

Ici, nous créons un objet de type `Object` avec l'instruction `new Object`. Cette nouvelle personne contient toutes les propriétés et méthodes du type `Object`, mais avec un nouveau nom. Vous pouvez ensuite ajouter vos propres propriétés et vos propres méthodes à l'objet `personne` de manière à décrire précisément ce qui est spécifique à une personne. Par exemple :

```
var personne = new Object();
personne.yeux = 2;
personne.oreilles = 2;
person.bras = 2;
person.mains = 2;
person.pieds = 2;
person.jambes = 2;
person.genre = "Homo sapiens";
```

Nous venons donc de définir les propriétés particulières à une personne *en général*. Mais imaginez maintenant que vous vouliez créer un nouvel objet pour un objet *en particulier*, disons Sandy Durand. Vous pourriez bien entendu simplement définir un objet supplémentaire appelé `SandyDurand`, et lui affecter les mêmes propriétés qu'à l'objet `personne`, plus celles qui sont éventuellement uniques dans le cas de Sandy Durand. Par exemple :

```
var SandyDurand = new Object();
SandyDurand.yeux = 2;
SandyDurand.oreilles = 2;
SandyDurand.bras = 2;
SandyDurand.mains = 2;
SandyDurand.pieds = 2;
SandyDurand.jambes = 2;
SandyDurand.genre = "Homo sapiens";
SandyDurand.profession = "Musicienne";
SandyDurand.habite = "Paris";
SandyDurand.cheveux = "Châtains";
SandyDurand.genreMusical = "Rock";
```

Pour autant, cette méthode est une énorme perte de temps et d'efficacité. Elle vous impose un tas de travail, et rien n'indique que Sandy Durand soit tout simplement une personne. Simplement, elle possède les mêmes propriétés qu'une personne, mais il pourrait tout aussi bien s'agir d'un robot ou de quoi que ce soit d'autre.

La solution consiste à créer un nouveau type d'objet, appelé `Personne`, puis à définir SandyDurand comme étant un objet de type `Personne`.



Notez que lorsque nous parlons du type d'un objet, nous mettons toujours en majuscule l'initiale de ce type. Cela n'a rien d'une obligation, mais c'est une convention presque universelle. Par exemple :

```
var personne = new Object ;
```

Ou bien :

```
var SandyDurand = new Personne () ;
```

Objets et constructeurs

Pour créer un nouveau type d'objet, vous définissez une nouvelle fonction, ou constructeur. Ces fonctions sont formées comme toutes les autres dans JavaScript, mais elles utilisent le mot-clé `this` pour affecter des propriétés à un objet. Le nouvel objet hérite ensuite des propriétés du type d'objet.

Voici par exemple un constructeur pour notre type d'objet Personne :

```
function Personne() {
this.yeux = 2;
this.oreilles = 2;
this.bras = 2;
this.mains = 2;
this.pieds = 2;
this.jambes = 2;
this.genre = "Homo sapiens";
```

Pour créer maintenant un nouvel objet de type Personne, il suffit d'affecter la fonction à une nouvelle variable. Par exemple :

```
Var SandyDurand = new Personne ()
```

L'objet `SandyDurand` hérite des propriétés du type d'objet `Personne`. Même si vous n'avez défini aucune propriété particulière pour l'objet `SandyDurand`, il contient déjà toutes celles de `Personne`.

Pour tester cette technique, exécutez le code du Listing 8.8 dans un navigateur Web.

LISTING 8.8 : Tester l'héritage.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
<title>Démo d'héritage</title>
</head>
<body>
  <script>

  function Person(){
    this.eyes = 2;
    this.ears = 2;
    this.arms = 2;
    this.hands = 2;
    this.feet = 2;
    this.legs = 2;
    this.species = "Homo sapiens";
  }
  var SandyDurand = new Person();
  alert("Sandy Durand a " + SandyDurand.feet + " pieds !");

  </script>
</body>
</html>
```

Le résultat de cette exécution est illustré sur la Figure 8.1.

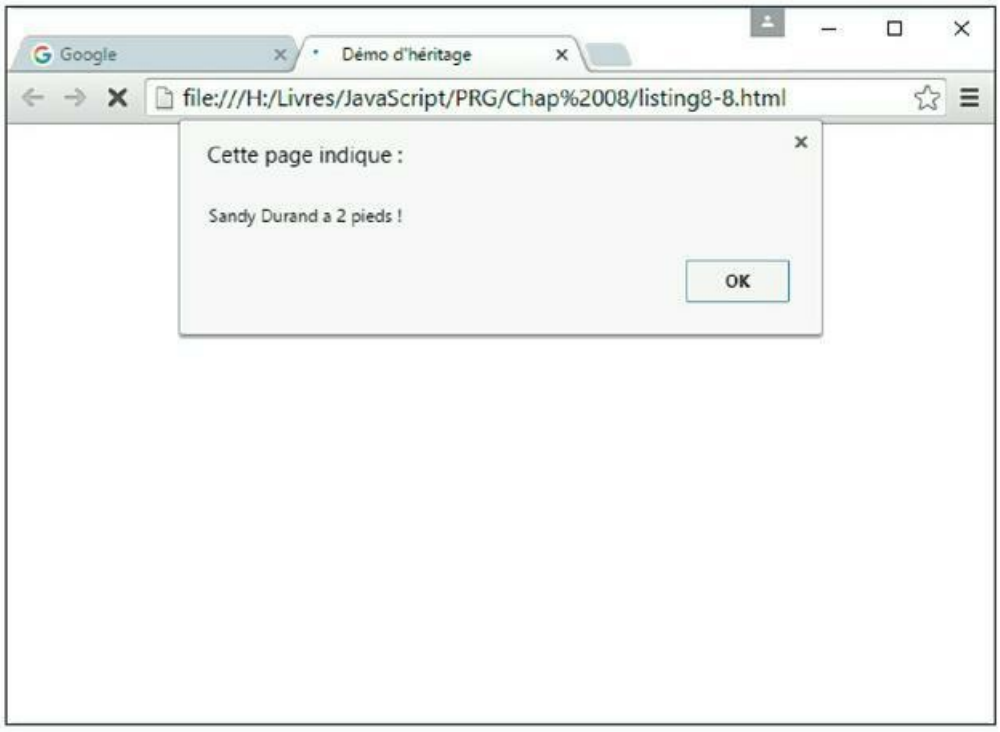


FIGURE 8.1 : Sandy Durand est une Personne.

Modifier un type d'objet

Supposons que vous avez votre type d'objet `Personne`, qui sert de prototype pour plusieurs autres objets. À un certain moment, vous réalisez que la personne, de même que tous les objets qui en héritent, devrait avoir quelques propriétés supplémentaires.

Pour modifier un prototype d'objet, vous utilisez la propriété `prototype` qui permet à chaque membre d'hériter des nouveautés. Le Listing 8.9 montre comment cela fonctionne.

LISTING 8.9 : Modifier un prototype d'objet.

```
function Personne(){
  this.eyes = 2;
  this.ears = 2;
  this.arms = 2;
  this.hands = 2;
  this.feet = 2;
  this.legs = 2;
  this.species = "Homo sapiens";
}

var SandyDurand = new Personne();
var LouisDupont = new Personne();
var JeanBon = new Personne();
// Personne a besoin de plus de propriétés !
Personne.prototype.knees = 2;
Personne.prototype.toes = 10;
Personne.prototype.elbows = 2;
// Vérifie les valeurs des nouvelles propriétés pour un objet existant
document.write (JeanBon.toes); // donne la valeur 10
```

Créer des objets avec Object.create

Une autre façon permettant de créer des objets à parti d'autres objets consiste à utiliser la méthode `Object.create`. L'avantage est ici que vous n'avez pas besoin de définir un constructeur. Cette méthode recopie simplement les propriétés d'un objet spécifié (le prototype) dans un autre objet (l'héritier).

Le Listing 8.10 illustre cette technique.

LISTING 8.10 : Utiliser Object.create pour créer un objet à partir d'un prototype.

```
// crée un objet Personne générique
var Personne = {
  eyes:  2,
  arms:  2,
  feet:  2
}

// crée l'objet SandyDurand, basé sur Personne
var SandyDurand = Object.create(Personne);

// teste une propriété héritée
document.write (SandyDurand.feet); // retourne la valeur 2
```

JavaScript sur le Web

DANS CETTE PARTIE...

Comprendre comment utiliser l'objet Window pour contrôler le navigateur

Manipuler les documents avec le DOM

Utiliser des événements dans JavaScript

Gérer les entrées et les sorties

Découvrir comment travailler avec CCS et les graphiques

Chapitre 9

Contrôler le navigateur avec l'objet Window

DANS CE CHAPITRE :

- **Comprendre le BOM (*Browser Object Model*)**
- **Ouvrir et fermer des fenêtres**
- **Obtenir les propriétés des fenêtres**
- **Redimensionner les fenêtres**

« L'astuce avec les théories, c'est de toujours garder une fenêtre ouverte de manière à pouvoir en jeter une si nécessaire. »

Bela Lugosi

Le modèle d'objet du navigateur, ou BOM (*Browser Object Model*), permet à JavaScript d'interagir avec les fonctionnalités du navigateur Web. Grâce à ce BOM, vous pouvez créer et redimensionner des fenêtres, afficher des messages d'avertissement, ou encore changer la page actuellement affichée dans le navigateur.

Dans ce chapitre, vous allez découvrir ce qui peut être fait dans la fenêtre du navigateur, et comment l'utiliser pour écrire de meilleurs programmes JavaScript.



N'oubliez pas de visiter le site Web compagnon de ce livre pour y télécharger les exemples de ce chapitre !

Comprendre l'environnement du navigateur

Les navigateurs Web sont des logiciels très complexes. Lorsqu'ils fonctionnent bien, ils opèrent en douceur et exécutent toutes leurs fonctionnalités sans heurts, procurant ainsi à l'utilisateur une expérience de navigation agréable. Mais nous savons tous que, dans le monde réel, il arrive que des problèmes surgissent, et même qu'un navigateur plante. Pour comprendre comment cela peut se produire, et donc comment mieux gérer les navigateurs, il est important d'en connaître les principaux composants, et comment ceux-ci interagissent.

L'interface utilisateur

La partie du navigateur avec laquelle vous interagissez lorsque vous tapez une adresse URL, cliquez sur le bouton Accueil, créez ou utilisez un favori, ou encore personnalisez les paramètres du navigateur, s'appelle l'*interface utilisateur*.

Cette interface utilisateur contient donc les menus, les cadres des fenêtres, les barres d'outils et les boutons qui se trouvent en dehors de la partie principale de la fenêtre du navigateur, là où se chargent les pages Web (voir la Figure 9.1).


Barre d'onglets
www.pourlesnuls.fr/la-collection.html
Barre d'adresse
Barre des favoris
Fenêtre de navigation
LE CATALOGUE
LA COLLECTION
POUR LES NULS
LES ÉVÈNEMENTS
L'ACTU DES NULS
Accueil > Histoire de la collection
La collection Pour Les Nuls a 15 ans déjà !
Le 17 janvier 2001, les Éditions First publient leurs premiers titres dans la collection « Pour les Nuls », puisés dans le catalogue de la déjà célèbre collection américaine « For Dummies ».
Quelques mois plus tard, en août 2001, c'est le premier titre créé en français qui est publié. Il s'agit du Français correct pour les Nuls écrit par un certain Jean-Joseph Julaud. Très attaché à la transmission du savoir et grand amoureux devant l'Éternel de notre langue, Jean-Joseph Julaud enseigne alors l'histoire-géographie et la littérature française dans un collège de Loire-Atlantique. Il ne sait pas encore qu'il va devenir l'emblème de cette collection atypique. En août 2004, il signe L'Histoire de France pour les Nuls et un peu aussi son destin ! Cet ouvrage s'est vendu, à ce jour, à plus d'un million d'exemplaires et devient l'un des livres consacré à l'histoire de France le plus vendu de ces dernières années. Un nouveau chapitre de la collection « Pour les Nuls » s'ouvre alors, et la collection s'impose comme la référence pour la transmission ludique des savoirs et des connaissances : avec les Nuls, tout devient enfin facile !
Janvier 2016, la désormais célèbre collection jaune et noire a séduit plus de 20 millions d'acheteurs francophones avec un choix d'environ 1 200 titres publiés aux Éditions First, tous sujets et formats confondus. Un succès d'autant plus important que la France reste le pays qui vend le plus de livres « Pour les Nuls » après les États-Unis et le Royaume-Uni.
15 ans POUR LES NULS
Outils pour développeurs

FIGURE 9.1 : Un navigateur et son interface utilisateur.

Le « loader »

On appelle *loader* la partie d'un navigateur qui communique avec les serveurs Web et télécharge les pages Web, les scripts, les fichiers CSS, les graphiques, et tous les autres composants qui constituent ces pages. En règle générale, c'est ce qui prend le plus de temps et provoque parfois l'incompréhension de l'utilisateur.

La *page HTML* est la première partie d'une page Web qui doit être téléchargée. Elle contient en particulier les liens et les scripts incorporés qui doivent être traités de manière à afficher la page.

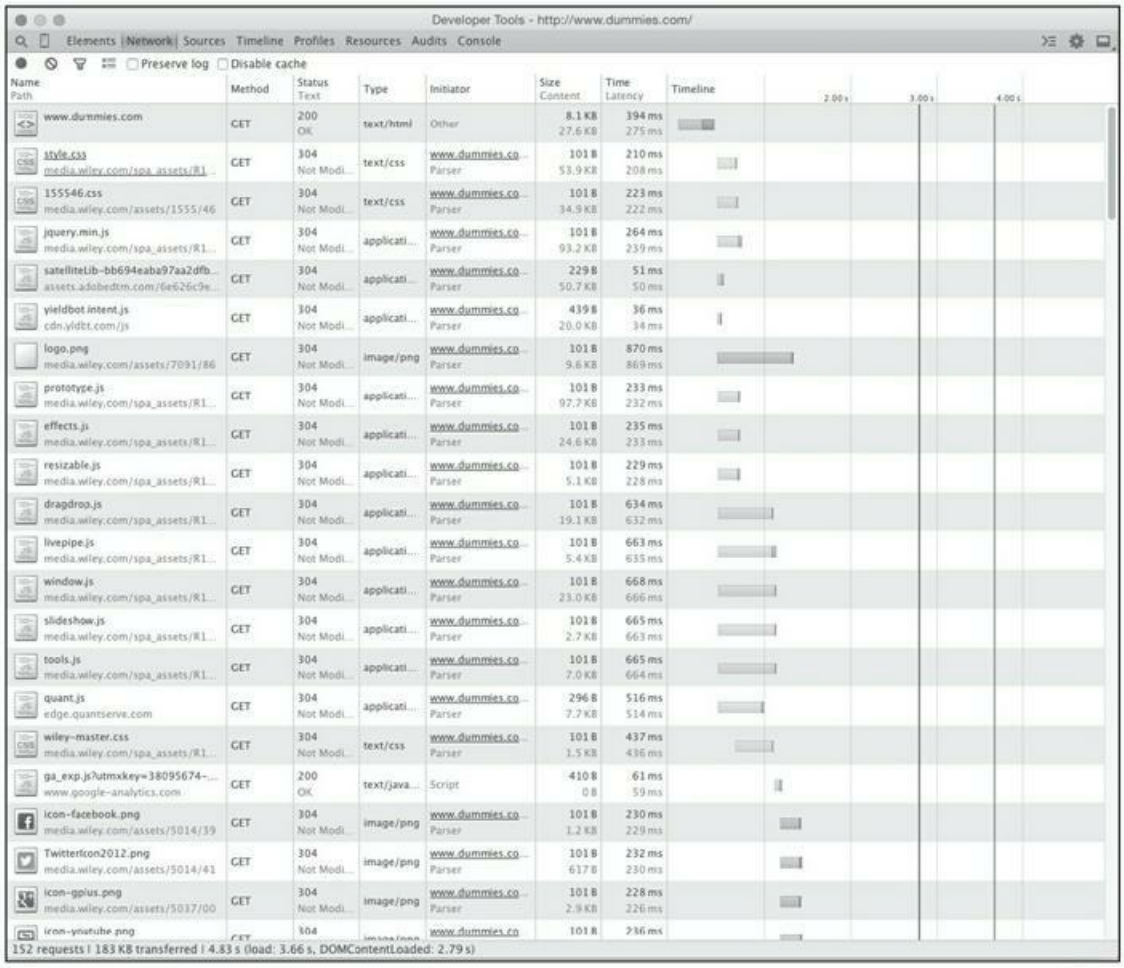


FIGURE 9.2 : Illustration des événements qui se produisent lors du téléchargement des composants d'un page Web.

La Figure 9.2 illustre l'onglet des outils de développement de Chrome. Il affiche une vue graphique de tout ce qui se passe depuis le chargement d'une page Web, de même qu'une ligne de temps (*timeline*) montrant la durée du téléchargement de chaque composant.

Une fois le document HTML téléchargé, le navigateur ouvre plusieurs connexions avec le serveur de manière à charger les autres parties de la page Web aussi rapidement que possible. En règle générale, les parties d'une page Web qui sont liées à cette page *via* le document HTML (ce que l'on appelle les *ressources*) sont chargées dans l'ordre où elles apparaissent dans ce document. Par exemple, un script placé dans la section `head` de la page sera chargé avant un autre script qui se trouverait vers la fin de celle-ci.



L'ordre de téléchargement des ressources est important pour l'efficacité du processus et la vitesse à laquelle la page Web peut être affichée dans la fenêtre du navigateur. De plus, pour que la page puisse être affichée correctement, il est indispensable que les styles CSS qui lui sont appliqués soient chargés et analysés tout de suite. C'est pourquoi ces styles CSS devraient toujours être placés en haut de la page, dans la section `head` du code HTML.

Même si, le plus souvent, le rôle de JavaScript concerne seulement les fonctionnalités de la page, il peut aussi arriver qu'il joue sur l'affichage de celle-ci. Dans ce cas, ce script devrait lui aussi être chargé dans l'en-tête du document HTML (après les styles CSS). Les scripts qui n'influent pas sur l'aspect de la page trouveront idéalement leur place tout à la fin du code HTML, juste avant la balise `</body>`. De cette manière, cela évite de provoquer des situations de blocage dans lesquelles le navigateur serait forcé d'attendre le chargement de ces scripts avant de pouvoir afficher quoi que ce soit, ce qui ne manquerait pas d'inquiéter ou d'irriter l'utilisateur.

Analyse du code HTML

Une fois une page Web téléchargée, le composant d'analyse HTML du navigateur va créer un modèle de cette page. C'est ce que l'on appelle le modèle d'objet du document (ou DOM, pour *Document Object Model*). Nous y

reviendrons en détail dans le Chapitre 10. Pour l'instant, il vous suffit de savoir que ce DOM est comme une cartographie de votre page Web. Les programmeurs JavaScript utilisent cette carte pour accéder aux différentes parties d'une page Web et les manipuler.

Lorsque cette phase d'analyse du code HTML est terminée, le navigateur commence à charger les autres composants de la page Web.

Analyse des styles CSS

Une fois les définitions des styles CSS d'une page Web totalement chargées, le navigateur se met à analyser ces styles et à déterminer lesquels s'appliquent au document HTML. Il s'agit d'un processus complexe, qui nécessite plusieurs passes pour appliquer ces styles correctement et pour prendre en compte leurs interactions.

Analyse du code JavaScript

L'étape suivante est l'analyse du code JavaScript. Au cours de ce processus, chaque script de votre page Web est compilé et exécuté, et ce dans l'ordre où il apparaît dans le document HTML. Si votre code JavaScript ajoute ou supprime des éléments, du texte ou des styles à l'intérieur du modèle DOM du document HTML, le navigateur ajustera en conséquence le code HTML et le rendu des styles CSS.

Mise en page et rendu

Finalement, une fois toutes les ressources de la page Web chargées et analysées, le navigateur détermine la manière dont il doit afficher la page, puis il affiche celle-ci. À moins que vous n'ayez spécifié qu'un script placé vers le début du document ne devrait être exécuté qu'à la fin du processus, le traitement et le rendu de vos scripts seront effectués dans l'ordre où ils sont inclus dans le document.



En général, il est préférable d'afficher une page aussi rapidement que possible, même si elle n'est pas encore totalement fonctionnelle. Les sites Web modernes utilisent cette stratégie (dite de *chargement différé*) pour améliorer la performance de leurs pages telle qu'elle est perçue par les utilisateurs. S'il vous est arrivé d'ouvrir une page Web et de devoir attendre un peu avant de pouvoir utiliser un formulaire ou un élément interactif, vous avez déjà vu ce chargement différé en action.

À propos du BOM

Les programmeurs JavaScript peuvent retrouver des informations sur le navigateur dont se sert un utilisateur, et contrôler des aspects de l'expérience utilisateur, *via* une API appelée le modèle d'objet du navigateur (ou BOM).

Il n'existe pas de standard officiel pour ce modèle. Différents navigateurs l'implémentent de différentes manières. Cependant, il existe certaines règles généralement acceptées concernant la manière dont JavaScript interagit avec les navigateurs Web.

L'objet Navigator

L'objet `Navigator` fournit à JavaScript un accès à des informations sur le navigateur de l'utilisateur. Cet objet tire son nom du premier navigateur Web à l'avoir implémenté, Netscape Navigator. Sachez que cet objet n'est pas un élément de JavaScript, mais une fonctionnalité des navigateurs Web à laquelle JavaScript est capable d'accéder. Tous les navigateurs Web modernes ont adopté la même terminologie pour faire référence à cet objet de haut niveau.

L'objet `Navigator` permet d'accéder à des informations utiles, comme :

- le nom du navigateur Web ;

- la version du navigateur Web ;
- l'emplacement physique de l'ordinateur sur lequel le navigateur est exécuté (si l'utilisateur autorise la géolocalisation de son appareil, bien sûr) ;
- la langue du navigateur ;
- le type d'ordinateur sur lequel le navigateur est exécuté.

Le Tableau 9.1 montre toutes les propriétés de cet objet `Navigator`.

Tableau 9.1 : Les propriétés de l'objet Navigator.

Propriété	Utilisation
`appCodeName`	Donne le nom de code du navigateur
`appName`	Donne le nom du navigateur
`appVersion`	Donne les informations de version du navigateur
`cookieEnabled`	Indique si les cookies sont actifs dans le navigateur
`geolocation`	Peut être utilisé pour obtenir la position physique de l'ordinateur de l'utilisateur
`language`	Donne la langue dans laquelle le navigateur est exécuté
`onLine`	Identifie si le navigateur est en ligne
`platform`	Donne la plate-forme pour laquelle le navigateur a été compilé
`product`	Donne le nom du moteur de navigation
`userAgent`	Donne le nom d'agent utilisateur renvoyé par le navigateur aux serveurs Web

Pour récupérer les propriétés de l'objet `Navigator`, vous utilisez la même syntaxe que pour n'importe quel autre objet, autrement dit soit la notation point, soit l'emploi de crochets droits. Le code du Listing 9.1, lorsqu'il est exécuté dans un navigateur Web, affiche toutes les propriétés et valeurs courantes de l'objet `Navigator`.

LISTING 9.1 : Les propriétés de l'objet Navigator et leurs valeurs.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
       <style>
       .columns {
        -webkit-column-count: 6; /* Chrome, Safari, Opera */
        -moz-column-count: 6; /* Firefox */
        column-count:  6;
        }
       </style>
  </head>
  <body>
       <div class="columns">
              <script>
              for (var prop in navigator){
                     document.write (prop + ": " + navigator[prop] + "
<br>");
              }
              </script>
       </div>
  </body>
 </html>
```

Le résultat est illustré sur la Figure 9.3.

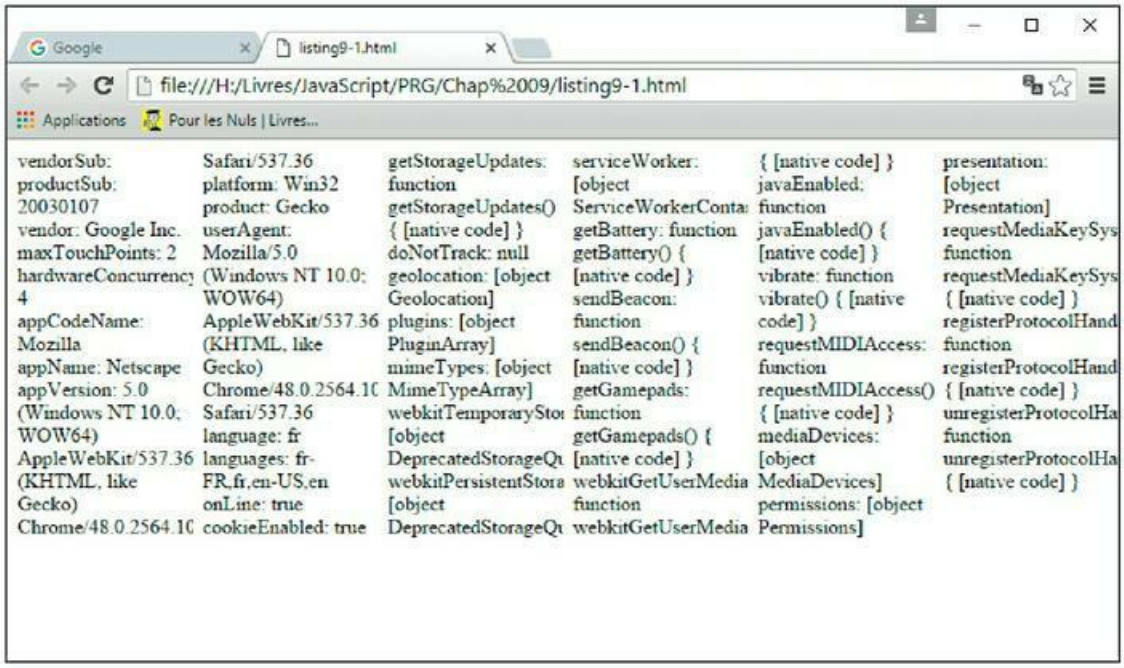

Google
listing9-1.html
file:///H:/Livres/JavaScript/PRG/Chap%2009/listing9-1.html
Applications
Pour les Nuls | Livres...
vendorSub: productSub: 20030107 vendor: Google Inc. maxTouchPoints: 2 hardwareConcurrency 4 appCodeName: Mozilla appName: Netscape appVersion: 5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.10
Safari/537.36 platform: Win32 product: Gecko userAgent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.10 Safari/537.36 language: fr languages: fr-FR,fr,en-US,en onLine: true cookieEnabled: true
getStorageUpdates: function getStorageUpdates() { [native code] } doNotTrack: null geolocation: [object Geolocation] plugins: [object PluginArray] mimeTypes: [object MimeTypeArray] webkitTemporaryStor [object DeprecatedStorageQu [native code] } webkitPersistentStora [object DeprecatedStorageQu
serviceWorker: [object ServiceWorkerConta getBattery: function getBattery() { [native code] } sendBeacon: function sendBeacon() { [native code] } getGamepads: function getGamepads() { [native code] } webkitGetUserMedia function webkitGetUserMedia
{ [native code] } javaEnabled: function javaEnabled() { [native code] } vibrate: function vibrate() { [native code] } requestMIDIAccess: function requestMIDIAccess() { [native code] } mediaDevices: [object MediaDevices] permissions: [object Permissions]
presentation: [object Presentation] requestMediaKeySys function requestMediaKeySys { [native code] } registerProtocolHand function registerProtocolHand { [native code] } unregisterProtocolHa function unregisterProtocolHa { [native code] }

FIGURE 9.3 : Les propriétés de l'objet Navigator et leurs valeurs.

Si vous exécutez le code du Listing 9.1, vous devriez pouvoir remarquer quelques étrangetés sur la sortie que vous obtenez. Par exemple, sur la Figure 9.3, la propriété `appName` indique Netscape alors que le navigateur utilisé est Google Chrome.

Cette curieuse valeur est une relique des temps anciens où les programmeurs utilisaient les propriétés de l'objet `Navigator` pour détecter si l'utilisateur se servait d'un navigateur particulier prenant en charge certaines fonctionnalités.

Lorsque de nouveaux navigateurs, comme Chrome ou Firefox, sont apparus, ils ont adopté comme valeur pour `appName` le nom historique, Netscape, pour s'assurer qu'ils étaient compatibles avec des sites Web qui détectaient des fonctionnalités de cette manière.



De nos jours, la détection des navigateurs n'est pas recommandée, et vous avez à votre disposition de bien meilleures méthodes pour déterminer si un navigateur prend en charge une fonctionnalité particulière que de regarder le contenu de la propriété `appName`. La méthode la plus courante consiste à examiner le modèle d'objet du document (DOM) pour rechercher les objets associés à la fonctionnalité que vous voulez utiliser. Par exemple, supposons que vous vouliez savoir si un navigateur prend en charge l'élément `audio` de HTML5. Vous pouvez alors effectuer le test suivant :

```
var test_audio= document.createElement("audio");
if (test_audio.play) {
console.log ("Le navigateur supporte l'audio HTML5");
} else {
console.log ("Le navigateur ne supporte pas l'audio HTML5");
}
```

L'objet Window

La partie principale d'un navigateur, celle qui affiche les pages Web, est appelée la *fenêtre* (*window*). C'est là que sont chargés les documents HTML et les ressources qui leur sont associées. Chaque onglet d'un navigateur Web est représenté en JavaScript par une instance de l'objet `Window`. Les propriétés de cet objet sont listées dans le Tableau 9.2.

Tableau 9.2 : Les propriétés de l'objet Window.

Propriété	Utilisation
closed	Valeur booléenne indiquant si une fenêtre a été fermée ou non
defaultStatus	Lit ou définit le texte affiché par défaut dans la barre d'état d'une fenêtre
document	Fait référence à l'objet `Document` de la fenêtre
frameElement	Obtient l'élément, comme <iframe> ou <object>, dans lequel la fenêtre est incorporée
frames	Liste tous les « sous-cadres » contenus dans la fenêtre courante
history	Obtient l'historique de navigation pour la fenêtre courante
innerHeight	Obtient la hauteur intérieure de la fenêtre courante
innerWidth	Obtient la largeur intérieure de la fenêtre courante
length	Obtient le nombre de cadres (*frames*) contenus dans la fenêtre

location	Obtient l'objet `Location` pour la fenêtre
name	Lit ou définit le nom de la fenêtre.
navigator	Obtient l'objet `Navigator` pour la fenêtre
opener	Obtient l'objet `Window` qui a créé la fenêtre courante
outerHeight	Obtient la hauteur extérieure de la fenêtre courante, y compris les barres de défilement et les barres d'outils
pageXOffset	Obtient le nombre de pixels à la suite d'un défilement horizontal dans la fenêtre
pageYOffset	Obtient le nombre de pixels à la suite d'un défilement vertical dans la fenêtre
parent	Fait référence au parent de la fenêtre courante
screen	Fait référence à l'objet `Screen` de la fenêtre
screenLeft	Obtient la distance horizontale en pixels entre le bord gauche de l'écran principal et le bord gauche de la fenêtre courante
screenTop	Obtient la distance verticale en pixels entre le bord supérieur de l'écran principal et le bord supérieur de la fenêtre courante
screenX	Obtient la coordonnée horizontale relativement à l'écran
screenY	Obtient la coordonnée verticale relativement à l'écran
self	Fait référence à la fenêtre courante
top	Fait référence à la fenêtre de navigation de niveau supérieur

Certaines des utilisations les plus courantes de ces propriétés sont les suivantes :

- Ouvrir un nouvel emplacement dans la fenêtre de navigation
- Trouver la taille d'une fenêtre de navigation
- Revenir à une page ouverte précédemment (c'est comme le bouton de retour arrière du navigateur)

Ouvrir une page Web avec la propriété window. location

Lire la valeur de `window.location` va renvoyer l'adresse URL de la page courante. Inversement, définir la valeur de cette propriété provoque le chargement de la page correspondant à cette nouvelle URL.

Le Listing 9.2 contient un script qui demande à l'utilisateur de saisir l'adresse d'une page Web. Cette page est ensuite chargée dans la fenêtre de navigation courante.

LISTING 9.2 : Charger une page Web en utilisant la propriété window.location.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
       <script>
      function loadNewPage (url){
        window.location = url;
      }
    </script>
  </head>
  <body>
       <script>
        var newURL = prompt("Entrez l'adresse d'une page web !");
        loadNewPage(newURL);
      </script>
  </body>
 </html>
```

La Figure 9.4 illustre le comportement du code du Listing 9.2.

Déterminer la taille d'une fenêtre de navigation

Lorsque vous concevez un site Web, ou une application Web, destinée à fonctionner sur plusieurs types de dispositifs (ce que l'on appelle aujourd'hui le *responsive design*), connaître la taille du navigateur Web, et en particulier sa largeur, est essentiel.

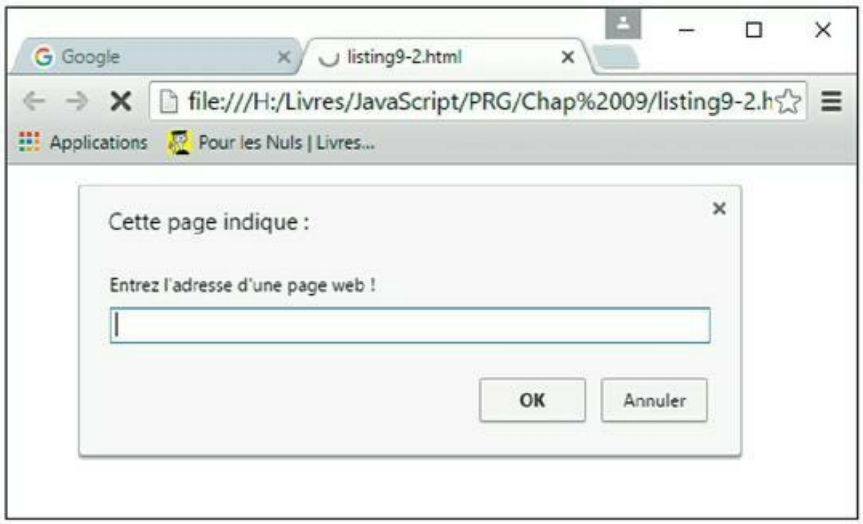


FIGURE 9.4 : La propriété window.location en action.

Les propriétés `window.innerWidth` et `window.innerHeight` vous fournissent cette information en pixels pour la fenêtre de navigation courante.



Il est aussi possible de faire appel à CSS pour déterminer ces informations, et c'est d'ailleurs une pratique assez courante. Cependant, il y a certaines différences dans la manière dont CSS et JavaScript traitent le cas des barres de défilement, ce qui peut influencer le choix de la technique à utiliser.

Le Listing 9.3 vous montre un exemple simple de responsive design en JavaScript. Exécutez ce code dans votre navigateur. Si la taille de la fenêtre est inférieure à 500 pixels, un certain message est affiché. Si elle est plus large, un autre message est émis.

LISTING 9.3 : Calculer la largeur de la fenêtre pour ajuster une page Web.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
    <title>Jouer avec la propriété window.innerWidth</title>
  </head>
  <body>
      <script>
        var currentWidth = window.innerWidth;
        if (currentWidth > 500) {
          document.write("<h1>Votre fenêtre est grande.</h1>");
        } else {
          document.write("<h1>Votre fenêtre est petite.</h1>");
        }
      </script>
  </body>
 </html>
```

Pour tester ce code, suivez ces étapes :

1. **Ouvrez un document HTML contenant le code du Listing 9.3 dans votre navigateur.**

   Si la largeur de la fenêtre est supérieure à 500 pixels, vous allez voir un message qui vous dit que votre fenêtre est grande.

2. **Redimensionnez la fenêtre du navigateur pour la rendre aussi étroite que possible.**

3. **Cliquez sur le bouton Rafraîchir du navigateur, ou appuyez sur Command + R (Mac) ou Ctrl + R (Windows) afin de recharger la page.**

   Remarquez que le message s'ajuste en fonction de la largeur de la fenêtre (voir la Figure 9.5).



FIGURE 9.5 : Un message différent s'affiche selon la largeur de la fenêtre.

Créer un bouton Retour en utilisant location et history

La propriété `history` de l'objet `Window` est une référence en lecture seule qui fournit l'historique de navigation de l'objet pour la fenêtre courante. L'utilisation de loin la plus courante de l'objet `History` est d'activer le bouton qui permet à l'utilisateur de revenir à une page déjà consultée.

Le Listing 9.4 illustre l'emploi de cette technique.

LISTING 9.4 : Implémenter un bouton Retour dans une application Web.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
    <title>Créer un bouton Retour</title>
    <script>
    function takeMeBack () {
      window.location(window.history.go(-1));
    }
    function getHistoryLength () {
      var l = window.history.length;
      return l;
    }
    </script>
  </head>
  <body>
      <script>
        var historyLength = getHistoryLength();
        document.write ("<p>Bienvenue ! Le nombre de pages que vous avez
visitées
        dans cette fenêtre est : " + historyLength + ".</p>");

      </script>
      <br>
      <a href="javascript:void(0);" onclick="takeMeBack();">Retour</a>
  </body>
 </html>
```

Pour voir ce code en action, suivez ces étapes :

1. **Ouvrez une fenêtre de navigation et visitez une page Web à votre convenance.**
2. **Toujours dans la même fenêtre, ouvrez un document HTML contenant le code du Listing 9.4.**
3. **Cliquez sur le lien Retour.**

   Vous revenez à la page précédente.

Si vous ouvrez le code du Listing 9.4 dans une fenêtre avant d'avoir ouvert une autre page Web, il ne va évidemment rien se passer. Le message vous indiquera tout simplement que votre historique est réduit à une page (celle que vous lisez), et le lien Retour sera inactif.



Utiliser les méthodes de l'objet Window

En plus de ses propriétés, l'objet `Window` possède plusieurs méthodes utiles, et donc que tout programmeur en JavaScript devrait connaître et utiliser. Le Tableau 9.3 fournit la liste complète de ces méthodes.

Tableau 9.3 : Les méthodes de l'objet Window.

Méthode	Utilisation
alert ()	Affiche une boîte d'alerte avec un message et le bouton OK
atob ()	Décode une chaîne encodée sur 64 bits
blur ()	Provoque la perte du focus pour la fenêtre courante
clearInterval ()	Annule le décompte de temps défini avec setInterval ()
clearTimeout ()	Annule le décompte de temps défini avec setTimeout ()
close ()	Referme la fenêtre ou la notification courante
confirm ()	Affiche une boîte de dialogue avec un message facultatif, ainsi que deux boutons : OK et Annuler
createPopup ()	Crée une fenêtre pop-up
focus ()	Donne le focus à la fenêtre courante
moveBy ()	Déplace la fenêtre courante du décalage spécifié
moveTo ()	Déplace une fenêtre vers la position spécifiée
open ()	Ouvre une nouvelle fenêtre
print ()	Imprime le contenu de la fenêtre courante
prompt ()	Affiche une boîte de dialogue demandant une saisie à l'utilisateur
resizeBy ()	Redimensionne la fenêtre d'un nombre spécifié de pixels
resizeTo ()	Redimensionne une fenêtre avec une hauteur et une largeur spécifiées
scrollBy ()	Fait défiler le document du décalage spécifié
scrollTo ()	Fait défiler le document vers un jeu de coordonnées spécifiques
setInterval ()	Appelle une fonction ou exécute une expression de manière répétée à un intervalle spécifié (en millisecondes)

setTimeout ()	Appelle une fonction ou exécute une expression après un intervalle spécifié (en millisecondes)
stop ()	Stoppe le chargement dans la fenêtre courante

Chapitre 10

Manipuler des documents avec le DOM

DANS CE CHAPITRE :

- **Comprendre le DOM (*Document Object Model*)**
- **Travailler avec les nœuds**
- **Se déplacer dans l'arbre**
- **Sélectionner des éléments**

« Aucun objet n'est mystérieux. Le mystère, ce sont vos yeux ».

Elizabeth Bowen

Comprendre le DOM est une question clé pour être capable de manipuler le texte ou le contenu HTML d'une page Web. Grâce au DOM, vous pouvez créer des animations, actualiser des données sans rafraîchir les pages, déplacer des objets dans le navigateur, et bien plus encore !



N'oubliez pas de visiter le site Web compagnon de ce livre pour y télécharger les exemples de ce chapitre !

Comprendre le DOM

Le DOM (*Document Object Model*) est l'interface grâce à laquelle JavaScript peut dialoguer et travailler avec les documents HTML à l'intérieur des fenêtres de navigation. Le DOM peut être visualisé sous la forme d'un arbre inversé, chaque partie du document HTML correspondant à une branche qui pointe vers son contenu.

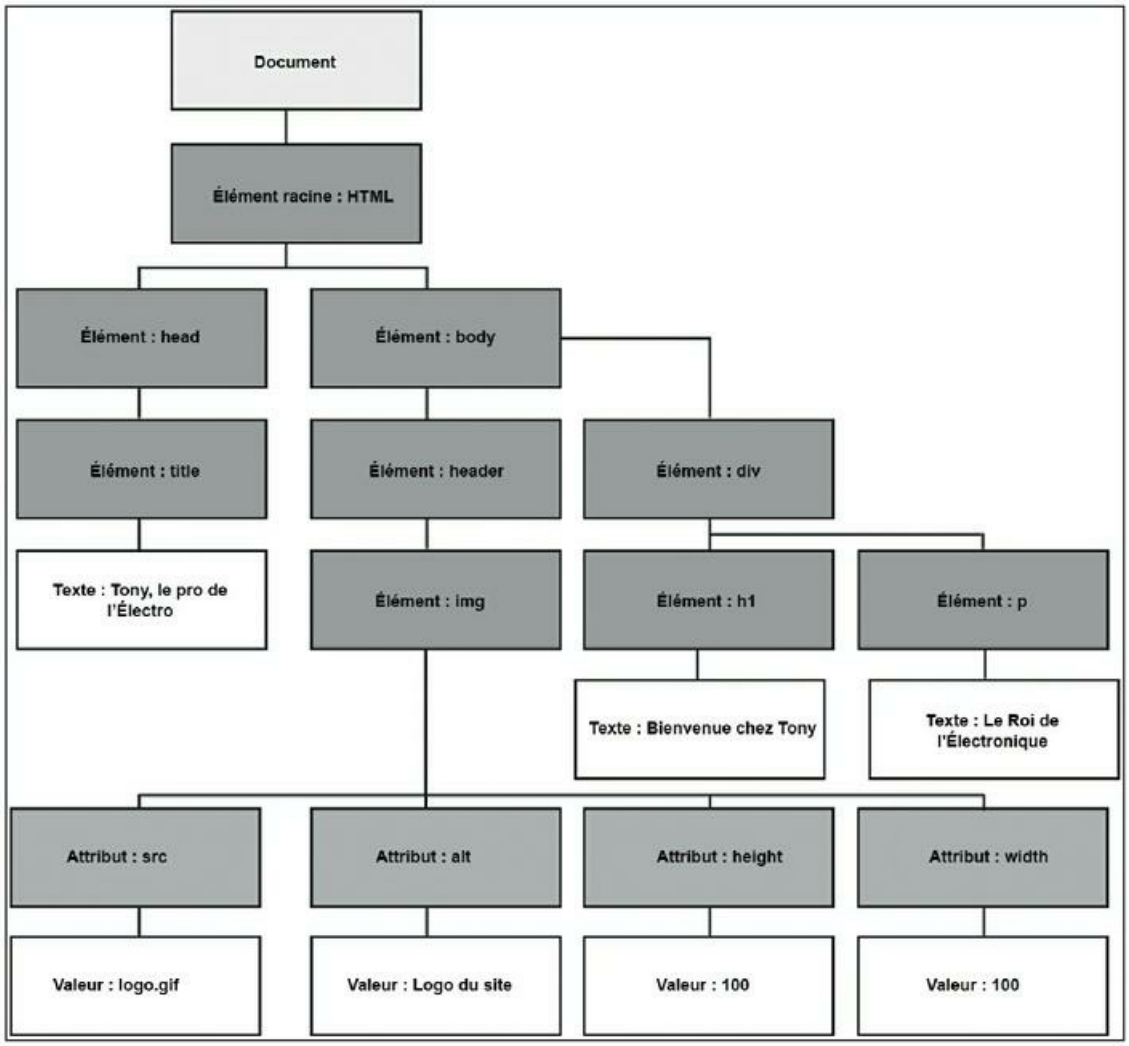


FIGURE 10.1 : Une représentation du modèle DOM pour le Listing 10.1.

Le Listing 10.1 illustre la structure d'une page Web. La Figure 10.1 montre la représentation du DOM correspondant.

LISTING 10.1 : Un document HTML.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
    <title>Tony, le Pro de l'Électro</title>
  </head>
  <body>
    <header>
      <img src="logo.gif" width="100" height="100" alt="Logo du site">
    </head>
    <div>
      <h1>Bienvenue chez Tony</h1>
      <p>Le Roi de l'Électronique</p>
    </div>
    <footer>
      copyright &copy; Tony
    </footer>
  </body>
</html>
```

Un arbre DOM est formé de composants individuels appelés *nœuds*. Le nœud principal, celui dont partent tous les autres, est le *nœud du document*. Le niveau immédiatement inférieur est le *nœud de l'élément racine*. Pour les documents HTML, ce nœud racine est HTML. Ensuite, chaque élément, attribut ou morceau de contenu est représenté par un nœud de l'arbre qui est issu d'un autre nœud.

Le modèle DOM possède plusieurs différents types de nœuds :

- **Document :** Le document HTML en entier est représenté dans ce nœud.
- **Éléments :** Les éléments du code HTML.
- **Attributs :** Les attributs associés à ces éléments.
- **Texte :** Le contenu textuel des éléments.
- **Commentaires :** Les commentaires insérés dans le document HTML.

Relations entre les nœuds

Les arbres DOM sont en fait de très proches parents des arbres généalogiques classiques. C'est d'ailleurs pourquoi on se sert d'une terminologie semblable :

- Chaque nœud, excepté le nœud racine, a un *parent*.
- Chaque nœud peut avoir un ou plusieurs *enfants*.
- Les nœuds issus du même parent sont dits frères (et/ou sœurs, évidemment, soit *sibling* en anglais). Ce parent est également appelé *ancêtre*, ou *ascendant*.

Du fait que les documents HTML possèdent souvent de multiples éléments de même type, le modèle DOM vous permet d'accéder à chacun individuellement en utilisant un système d'indexation. Par exemple, vous pouvez faire référence au premier élément `<p>` d'un document sous la forme `p[0]`, `p[1]` représentant le second et ainsi de suite.



Bien que cette notation des nœuds puisse faire penser à celle d'un tableau, ce n'est pas le cas. Vous pouvez parcourir dans une boucle le contenu d'une liste de nœuds, mais les méthodes des tableaux ne s'y appliquent pas.

Sur le Listing 10.2, les trois éléments `<p>` sont tous des enfants de l'élément `<div>`. Du fait qu'ils ont le même parent, ils appartiennent donc à la même fratrie.



Dans le Listing 10.2, les commentaires HTML sont également des enfants de l'élément `section`. Le dernier commentaire avant la balise de fermeture de la section est appelé le *dernier enfant* de cette section.

En comprenant les relations entre les nœuds d'un document, vous pourrez vous servir de l'arbre DOM pour retrouver n'importe quel élément de ce document.

LISTING 10.2 : Les relations parents, enfants, et frères et sœurs, dans un document HTML.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
    <title>La famille HTML</title>
  </head>
  <body>
    <section> <!-- parent des 3 éléments suivants, enfant de body -->
      <p>Premier</p> <!-- premier enfant de l'élément section,
frère/soeur de 2
        autres éléments <p> -->
      <p>Second</p> <!-- second enfant de l'élément section, frère/soeur
de 2
        autres éléments <p> -->
      <p>Troisième</p> <!-- troisième enfant de l'élément section,
frère/soeur de
2 autres éléments <p> -->
    </section>
  </body>
</html>
```

Le Listing 10.3 est un document HTML qui contient un script affichant tous les nœuds enfants de l'élément `section`.

LISTING 10.3 : Afficher les nœuds enfants de l'élément section.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
    <title>La famille HTML</title>
  </head>
  <body>
    <section> <!-- parent des 3 éléments suivants, enfant de body -->
      <p>Premier</p> <!-- premier enfant de l'élément section,
frère/soeur de 2
       autres éléments <p> -->
      <p>Second</p> <!-- second enfant de l'élément section, frère/soeur
de 2
       autres éléments <p> -->
      <p>Troisième</p> <!-- troisième enfant de l'élément section,
frère/soeur de 2
       autres éléments <p> -->
    </section>
    <h1>Noeuds dans l'élément section</h1>
    <script>
      var myNodelist = document.body.childNodes[1].childNodes;
      for (i = 0; i < myNodelist.length; i++){
        document.write (myNodelist[i] + "<br>");
      }
    </script>
  </body>
</html>
```

La Figure 10.2 illustre la sortie produite par le Listing 10.3 dans un navigateur. Remarquez que le premier nœud enfant de l'élément `section` est de type texte. Si vous observez de près les balises HTML du Listing 10.3, vous constaterez qu'il y a une unique espace entre la balise ouvrante `<section>` et le commentaire. Même quelque chose d'aussi simple que cette espace produit un nœud entier dans le modèle DOM. Ce fait doit être pris en considération lorsque vous naviguez dans le DOM en utilisant les relations entre les nœuds.

Le modèle DOM de HTML fournit également divers mots-clés permettant de naviguer entre les nœuds en utilisant leur position relative par rapport à leurs frères et sœurs et à leurs parents. Ces propriétés relatives sont les suivantes :

- `firstChild` : Fait référence au premier enfant du nœud.
- `lastChild` : Fait référence au dernier enfant du nœud.
- `nextSibling` : Fait référence au prochain nœud ayant le même nœud parent.
- `previousSibling` : Fait référence au nœud précédent ayant le même nœud parent.

Le Listing 10.4 montre comment utiliser ces propriétés relatives pour traverser l'arborescence du modèle DOM.

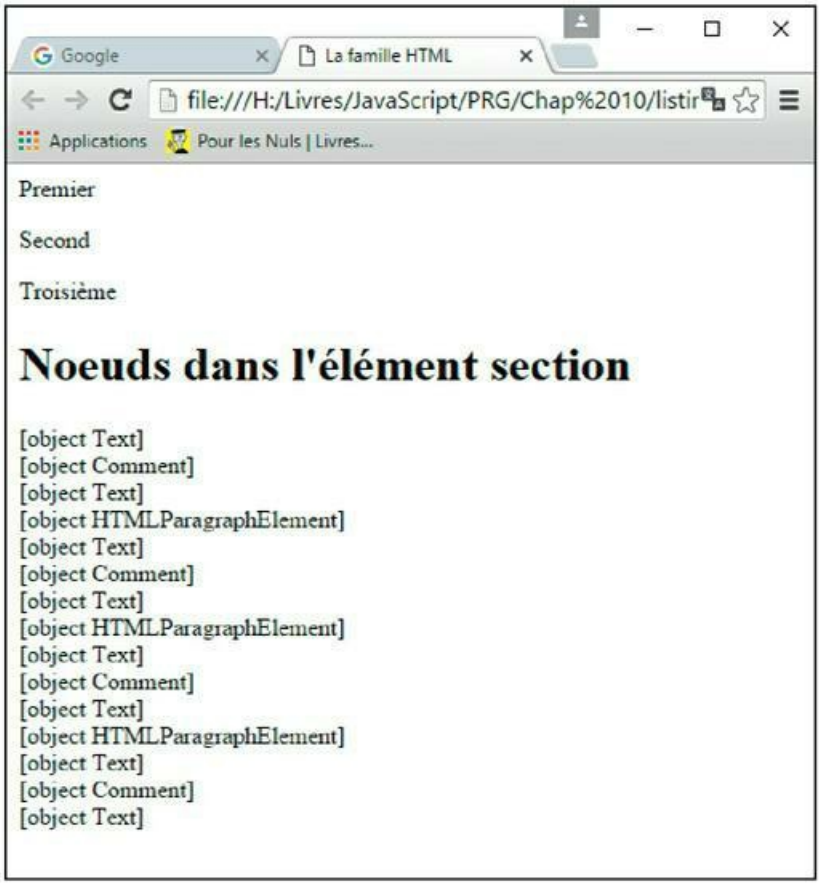


FIGURE 10.2 : Visualiser la sortie produite par le Listing 10.3.

LISTING 10.4 : Utiliser firstChild et lastChild pour mettre en valeur des liens de navigation.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
    <title>Les Iguanes ne sont pas sympathiques</title>
    <script>
    function boldFirstAndLastNav() {
      document.body.childNodes[1].firstChild.style.fontWeight="bold";
      document.body.childNodes[1].lastChild.style.fontWeight="bold";
    }
    </script>

  </head>
  <body>
    <nav><a href="home.html">Accueil</a> | <a href="why.html">Pourquoi
les
Iguanes ne sont pas sympathiques ?</a> | <a href="what.html">Que faire ?
</a> | <a
href="contact.html">Contactez nous</a></nav>
    <p>Pour le découvrir, visitez l'Amazonie. Utilisez les liens ci-
dessus pour
en savoir plus.</p>
    <script>
      boldFirstAndLastNav();
    </script>
  </body>
</html>
```

Remarquez sur le Listing 10.4 que tous les espacements entre les éléments de la section <nav> doivent être supprimés de manière à ce que les propriétés `firstChild` et `lastChild` puissent accéder aux liens que nous voulons sélectionner pour en changer le style.

La Figure 10.3 illustre le résultat produit par ce code dans un navigateur. Vous pouvez constater que seuls le premier et le dernier lien sont mis en caractères gras.


Nouvel onglet
Les Iguanes ne sont pas sy
file:///H:/Livres/JavaScript/PRG/Chap%2010/listing10-4.html
Applications
Pour les Nuls | Livres...
Accueil | Pourquoi les Iguanes ne sont pas sympathiques ? | Que faire ? | Contactez nous
Pour le découvrir, visitez l'Amazonie. Utilisez les liens ci-dessus pour en savoir plus.

FIGURE 10.3 : La sortie produite par le code du Listing 10.4 dans un navigateur Web.

Il s'agit du premier exemple dans lequel nous utilisons le modèle DOM pour modifier des éléments existants à l'intérieur du document. Cependant, cette méthode de sélection n'est pratiquement jamais employée. Elle est trop sujette à des erreurs, et trop difficile à interpréter et à utiliser.

Dans la section qui suit, nous verrons que le DOM nous propose de bien meilleurs procédés pour traverser et manipuler son arborescence que de dénombrer ses enfants.

Utiliser les propriétés et les méthodes de l'objet Document

L'objet Document fournit des propriétés et des méthodes pour travailler avec les documents HTML. Le Tableau 10.1 fournit la liste complète des propriétés de l'objet Document, et le Tableau 10.2 celle de ses méthodes.

Tableau 10.1 : Propriétés de l'objet Document.

Propriété	Utilisation
`anchors`	Récupère une liste de toutes les ancres (éléments <a> avec des noms d'attributs) du document
`applets`	Récupère une liste ordonnée de toutes les applets du document
`baseURI`	Récupère l'URI de base du document
`body`	Récupère le nœud <body> ou <frameset> dans le corps du document
`cookie`	Récupère ou définit des couples nom/valeur de cookies dans le document
`doctype`	Récupère la déclaration de type associée avec le document
`documentElement`	Récupère l'élément qui constitue la racine du document (par exemple, l'élément <html> dans un document HTML)
`documentMode`	Récupère le mode utilisé par le navigateur pour effectuer le rendu du document
`documentURI`	Récupère ou définit la localisation du document

`domain`	Récupère le nom de domaine du serveur qui a chargé le document
`embeds`	Récupère une liste de tous les éléments `<embed>` du document
`forms`	Récupère une collection de tous les éléments `<form>` du document
`head`	Récupère l'élément `<head>` du document
images	Récupère une liste de tous les éléments `<img>` du document
`implementation`	Récupère l'objet `DOMImplementation` qui gère le document
`lastModified`	Récupère la date et l'heure de la dernière modification du document
`links`	Récupère une collection de tous les éléments `<area>` et `<a>` du document qui contiennent l'attribut `href`
`readyState`	Récupère l'état de chargement du document. Renvoie `loading` en cours de chargement, `interactive` lorsque l'analyse du contenu est achevée, et `complete` lorsque le chargement est totalement terminé
`referrer`	Obtient l'URL de la page depuis laquelle le document courant était lié
`scripts`	Récupère une liste des éléments `<scripts>` du document
`title`	Récupère ou définit le titre du document
URL	Récupère l'URL complète du document

Tableau 10.2 : Les méthodes de l'objet Document.

Méthode	Utilisation
`addEventListener ()`	Assigne un gestionnaire d'événement au document
`adoptNode ()`	Adopte un nœud à partir d'un document externe
`close ()`	Termine le flux d'écriture en sortie du document qui a été ouvert avec `document.open ()`
`createAttribute ()`	Créé=e un nœud d'attribut
`createComment ()`	Crée un nœud de commentaire
`createDocumentFragment ()`	Crée un fragment de document vide
`createElement ()`	Crée un nœud d'élément
`createTextNode ()`	Crée un nœud de texte
`getElementById ()`	Récupère l'élément ayant l'attribut ID spécifié
`getElementByClassName ()`	Récupère tous les éléments possédant le nom de classe spécifié

`getElementByName ()`	Récupère tous les éléments ayant le nom spécifié
`getElementsByTagName ()`	Récupère tous les éléments ayant le nom de balise spécifié
`importNode ()`	Copie et importe un nœud provenant d'un document externe
`normalize ()`	Efface les nœuds de texte vides et joint les nœuds adjacents
`open ()`	Ouvre un document en écriture
`querySelector ()`	Récupère le premier élément qui correspond au groupe de sélecteurs spécifié dans le document
`querySelectorAll ()`	Récupère une liste de tous les éléments qui correspondent au groupe de sélecteurs spécifié dans le document
`removeEventListener ()`	Efface un gestionnaire d'événement qui a été ajouté au document en utilisant la méthode addEventListener ()
`renameNode ()`	Renomme un nœud existant
write ()	Écrit du code JavaScript code ou des expressions HTML dans un document
`writeIn ()`	Écrit du code JavaScript code ou des expressions HTML dans un document et passe à la ligne après chaque instruction

Utiliser les propriétés et les méthodes de l'objet Element

L'objet `Element` fournit des propriétés et des méthodes pour travailler avec les éléments HTML d'un document. Le Tableau 10.3 liste toutes les propriétés de l'objet Element, ses méthodes étant décrites dans le Tableau 10.4.

Tableau 10.3 : Les propriétés de l'objet Element.

Méthode	Utilisation
`accessKey`	Récupère ou définit l'attribut `accesskey` de l'élément
`attributes`	Récupère une collection de tous les attributs de l'élément enregistrés pour le nœud spécifié (retourne un `NameNodeMap`)
`childElementCount`	Récupère le nombre d'éléments enfants dans le nœud spécifié
`childNodes`	Récupère une liste des nœuds enfants de l'élément
`children`	Récupère une liste des éléments enfants de l'élément
`classList`	Récupère le ou les noms de classe de l'élément
`className`	Récupère ou définit la valeur de l'attribut de classe de l'élément

`clientHeight`	Obtient la hauteur interne de l'élément, y compris la valeur de remplissage
`clientLeft`	Obtient la largeur de la bordure gauche de l'élément
`clientTop`	Obtient la largeur de la bordure supérieure de l'élément
`clientWidth`	Obtient la largeur de l'élément, y compris la valeur de remplissage
`contentEditable`	Lit ou définit la valeur de l'attribut « éditable » de l'élément
`dir`	Lit ou définit la valeur de l'attribut `dir` de l'élément
`firstChild`	Récupère le nœud du premier enfant de l'élément
`firstElementChild`	Récupère le premier enfant de l'élément
`id`	Lit ou définit la valeur de l'attribut id de l'élément
`innerHTML`	Récupère ou définit le contenu de l'élément
`isContentEditable`	Retourne *true* si le contenu d'un élément est éditable, `false`.
lang	Lit ou définit la langue de base pour l'attribut des éléments
`lastChild`	Récupère le dernier nœud enfant de l'élément
`lastElementChild`	Récupère le dernier enfant de l'élément
`namespaceURI`	Obtient le nom d'espace URI pour le premier nœud de l'élément
nextSibling	Obtient le nœud suivant situé au même niveau
`nextElementSibling`	Obtient l'élément suivant situé au même niveau de nœud
`nodeName`	Récupère le nom du nœud courant
`nodeType`	Obtient le type du nœud courant
`nodeValue`	Lit ou définit la valeur du nœud
`offsetHeight`	Récupère la hauteur de l'élément, y compris le remplissage vertical, l'encadrement et la barre de défilement
`offsetWidth`	Récupère la largeur de l'élément, y compris le remplissage horizontal, l'encadrement et la barre de défilement
`offsetLeft`	Obtient la position du décalage horizontal de l'élément
`offsetParent`	Obtient le décalage du conteneur de l'élément
`offsetTop`	Obtient la position du décalage vertical de l'élément
`ownerDocument`	Obtient l'élément racine (le nœud document) pour un élément
`parentNode`	Récupère le nœud parent de l'élément
`parentElement`	Récupère l'élément parent de l'élément

`previousSibling`	Obtient le nœud précédent situé au même niveau
`previousElementSibling`	Obtient l'élément précédent situé au même niveau de nœud
`scrollHeight`	Obtient la hauteur totale de l'élément, y compris le remplissage
`scrollLeft`	Obtient ou définit le nombre de pixels du défilement horizontal de l'élément
`scrollTop`	Obtient ou définit le nombre de pixels du défilement vertical de l'élément
`scrollWidth`	Obtient la largeur totale de l'élément, y compris le remplissage
`style`	Récupère ou définit la valeur de l'attribut `style` de l'élément
`tabIndex`	Récupère ou définit la valeur de l'attribut `tabindex` de l'élément
`tagName`	Obtient le nom de balise de l'élément
`textContent`	Récupère ou définit le contenu textuel du nœud et de ses descendants
`title`	Récupère ou définit la valeur de l'attribut `title` de l'élément
`length`	Récupère le nombre de nœuds mémorisé dans NodeList

Tableau 10.4 : Les méthodes de l'objet Element.

Méthode	Utilisation
`addEventListener ()`	Enregistre un gestionnaire d'événement dans l'élément
`appendChild ()`	Insère un nouveau nœud enfant dans l'élément (en dernière position)
`blur ()`	Fait perdre le focus à l'élément
`click ()`	Simule un clic de souris sur l'élément
`cloneNode ()`	Clone l'élément
`compareDocumentPosition ()`	Compare la position de deux éléments dans le document
`contains ()`	Génère `true` si le nœud est un descendant d'un autre nœud, `false` sinon
`focus ()`	Donne le focus à l'élément
`getAttribute ()`	Récupère la valeur de l'attribut spécifié pour le nœud de l'élément
`getAttributeNode ()`	Récupère le nœud de l'attribut spécifié
`getElementsByClassName ()`	Récupère une collection de tous les éléments enfants possédant le nom de classe spécifié
`getElementByTagName ()`	Récupère une collection de tous les éléments enfants possédant le nom de balise spécifié

getFeature ()	Obtient un objet qui implémente l'API's de la fonctionnalité spécifiée
`hasAttribute ()`	Génère `true` si l'élément possède l'attribut spécifié, `false` sinon
`hasAttributes ()`	Génère `true` si l'élément possède des attributs, `false` sinon
`hasChildNodes ()`	Génère `true` si l'élément possède des nœuds enfants, `false` sinon
`insertBefore ()`	Entre un nouveau nœud enfant avant le nœud existant spécifié
`isDefaultNamespace ()`	Génère `true` si le `namespaceURI` spécifié est celui qui est pris par défaut, `false` sinon
`isEqualNode ()`	Évalue l'égalité (ou non) de deux éléments
`isSameNode ()`	Compare deux éléments pour savoir s'ils se trouvent dans le même nœud
`isSupported ()`	Génère `true` si la fonctionnalité spécifiée est supportée par l'élément
`normalize ()`	Joint les nœuds spécifiés et leurs nœuds adjacents, et supprime les nœuds de texte vides
querySelector ()	Obtient le premier élément enfant correspondant au(x) sélecteur(s) CSS spécifié(s) de l'élément
`querySelectorAll ()`	Obtient tous les éléments enfants correspondant au(x) sélecteur(s) CSS spécifié(s) de l'élément
`removeAttribute ()`	Supprime l'attribut spécifié de l'élément
`removeAttributeNode ()`	Supprime le nœud d'attribut spécifié de l'élément et retrouve le nœud supprimé
`removeChild ()`	Supprime le nœud enfant spécifié
`replaceChild ()`	Remplace le nœud enfant spécifié par un autre
`removeEventListener ()`	Supprime le gestionnaire d'événement spécifié
`setAttribute ()`	Modifie ou définit l'attribut spécifié par la valeur indiquée
`setAttributeNode ()`	Modifie ou définit le nœud d'attribut spécifié
`toString ()`	Change un élément en chaîne
`item ()`	Récupère le nœud ayant l'indice spécifié dans NodeList

Travailler avec le contenu des éléments

Vous pouvez afficher le type des nœuds et les valeurs de ceux-ci en utilisant le modèle DOM de HTML. Vous avez également la possibilité de définir la valeur des propriétés des éléments du DOM en vous servant de l'objet `Element`. Lorsque vous utilisez JavaScript pour définir ces propriétés, les nouvelles valeurs sont prises en compte en temps réel dans le document HTML.

Modifier les propriétés des éléments dans un document Web de manière à refléter ce changement immédiatement dans le navigateur, sans qu'il soit besoin de rafraîchir ou de recharger la page, est une des pierres angulaires de ce que l'on a pris l'habitude d'appeler le Web 2.0.

innerHTML

La plus importante propriété que vous puissiez modifier *via* le DOM est `innerHTML`.

La propriété `innerHTML` d'un élément contient tout ce qui se trouve entre la balise de début et la balise de fin de cet élément. Par exemple, dans le code qui suit, la propriété `innerHTML` de l'élément `div` contient un élément `p` et son nœud enfant de type texte :

```
<body><div><p>Voici un exemple de texte.</p></div></body>
```



Il est très courant dans la programmation Web de créer des éléments `div` vides dans un document HTML, puis d'utiliser la propriété `innerHTML` pour y insérer dynamiquement du texte ou du code HTML.

Pour récupérer ou définir la valeur de la propriété `innerHTML`, vous pouvez par exemple utiliser le code suivant :

```
var getTheInner = document.body.firstChild.innerHTML;
document.write (getTheInner);
```

Ici, la valeur qui sera produite par la méthode `document.write` sera :

```
<p>Voici un exemple de texte.</p>
```

Définir la propriété `innerHTML` s'effectue exactement comme s'il s'agissait d'une quelconque propriété d'un quelconque élément :

```
document.body.firstChild.innerHTML = "Hello tous ! " ;
```

Le résultat de ce code sera que l'élément `p` ainsi que la phrase qui était contenu dans le document d'origine seront remplacés par les mots « Hello tous ! ». Le document original, tel qu'il est enregistré, n'est pas modifié, mais sa représentation selon le modèle DOM ainsi que le rendu de la page Web dans le navigateur vont être actualisés pour refléter la nouvelle valeur. Du fait que la représentation DOM du document HTML est ce qui est affiché par le navigateur, l'aspect de votre page Web dans celui-ci changera également.

Définir des attributs

Pour définir la valeur d'un attribut HTML, vous pouvez faire appel à la méthode `setAttribute` () :

```
document.body.firstChild.innerHTML.setAttribute ("classe", "maClasse") ;
```

Le résultat de cette instruction est que le premier élément enfant de l'élément `body` prend un nouvel attribut appelé `classe` avec une valeur égale à `maClasse`.

Trouver des éléments par ID, nom de balise ou classe

Les méthodes `getElementBy` fournissent un accès facile à tout élément, ou groupe d'éléments, dans un document sans avoir à se reposer sur les relations parent/enfant des nœuds. Les trois méthodes les plus utilisées sont :

- `getElementByID`
- `getElementByTagName`
- `getElementByClassName`

getElementByID

C'est de très loin la méthode la plus employée pour sélectionner des éléments. `getElementByID` est essentielle pour le développement d'applications Web modernes. Avec cet outil pratique, vous pouvez trouver n'importe quel élément avec lequel vous avez besoin de travailler, simplement en faisant référence à un attribut `id` unique. Quoi qu'il se passe d'autre dans le document HTML, `getElementByID` sera toujours à votre disposition pour sélectionner exactement l'élément voulu.

Le Listing 10.5 illustre la puissance de `getElementByID` pour vous permettre de rassembler tout votre code JavaScript dans votre document, ou encore pour le rendre modulaire. Grâce à cette méthode, il vous est possible de travailler avec n'importe quel élément, où qu'il se trouve dans votre document, dès lors que vous connaissez son `id`.

LISTING 10.5 : Utiliser getElementByID pour sélectionner des éléments.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
    <title></title>
    <script>
    function calculateKPL(kms,litres){
      document.getElementById("displayKms").innerHTML = parseInt(kms);
      document.getElementById("displayLitres").innerHTML =
parseInt(litres);
      document.getElementById("displayKPL").innerHTML = kms/litres;
    }
    </script>
  </head>
  <body>
    <p>Vous avez conduit <span id="displayKms">___</span> kilomètres.</p>
    <p>Vous avez consommé <span id="displayLitres">___</span> litres de
       carburant.</p>
    <p>Distance moyenne parcourue par litre de carburant : <span
id="displayKPL">
       ___</span> km.
    <script>
      var kmsParcourus = prompt("Entrez la distance parcourue");
      var litresCarb = prompt("Entrez le nombre de litres de carburant
utilisés");
      calculateKPL(kmsParcourus,litresCarb);
    </script>
  </body>
</html>
```

getElementByTagName

La méthode `getElementByTagName` retourne une liste de tous les éléments possédant le nom de balise spécifié. Sur le Listing 10.6, par exemple, cette méthode est utilisée pour sélectionner tous les éléments `h1` et changer leur propriété `innerHTML` en une suite de valeurs séquentielles.

LISTING 10.6 : Utiliser getElementByTagName pour sélectionner et modifier des éléments.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
    <title></title>
    <script>
    function numberElements(tagName){
      var getTags = document.getElementsByTagName(tagName);
      for(i=0; i < getTags.length; i++){
        getTags[i].innerHTML = i+1;
      }
    }
    </script>
  </head>
  <body>
    <h1>ce texte va disparaître</h1>
    <h1>celui-ci sera remplacé</h1>
    <h1>JavaScript va effacer ceci</h1>
    <script>
      numberElements("h1");
    </script>
  </body>
</html>
```

getElementByClassName

Cette méthode est semblable en termes de fonctionnement à `getElementByTagName`, à ceci près qu'elle utilise les valeurs de l'attribut `class` pour sélectionner les éléments. Sur l'exemple du Listing 10.7, la fonction recherche les éléments ayant pour classe `error` et change la valeur de leur propriété `innerHTML`.

LISTING 10.7 : Utiliser getElementByClassName pour sélectionner et modifier des éléments.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
    <title></title>
    <script>
    function checkMath(result){
      var userMath = document.getElementById("answer1").value;
      var errors = document.getElementsByClassName("error");
      if(parseInt(userMath) != parseInt(result)) {
        errors[0].innerHTML = "C'est faux. Vous avez entré " + userMath +
". La
          bonne réponse est " + result + ".";
      } else {
        errors[0].innerHTML = "Correct !";
      }
    }
    </script>
  </head>
  <body>
    <label for = "number1">4+1 = </label><input type="text" id="answer1"
value="">
    <button id="submit" onclick="checkMath(4+1);">Révisez vos maths !
</button>
    <h1 class="error"></h1>
  </body>
</html>
```



Remarquez que le code du Listing 10.7 utilise un attribut `onclick` à l'intérieur de l'élément `button`. Il s'agit d'un exemple de gestionnaire d'événement DOM. Les gestionnaires d'événements sont traités dans le Chapitre 11.

La Figure 10.4 illustre le comportement de ce code si l'utilisateur entre une mauvaise réponse (si, ça peut arriver) à gauche, et une bonne réponse à droite.

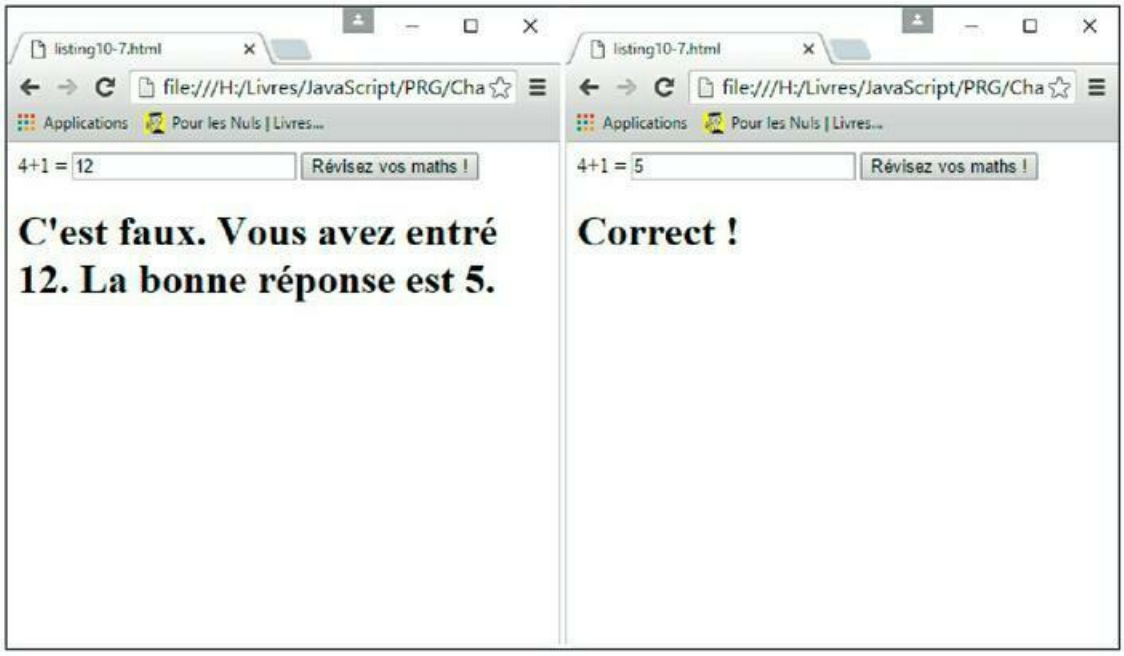


FIGURE 10.4 : Avec getElementByClassName, il est possible de sélectionner un élément pour afficher un message d'erreur (ou de réussite).

Utiliser les propriétés de l'objet Attribut

L'objet `Attribut` fournit des propriétés pour travailler avec les attributs des éléments HTML. Le Tableau 10.5 liste toutes ces propriétés.

Tableau 10.5 : Les propriétés de l'objet Attribut.

Propriété	Utilisation
`isId`	Génère `true` si l'attribut est un id, `false` sinon
`name`	Fournit le nom de l'attribut
`value`	Récupère ou définit la valeur de l'attribut
`specified`	Génère `true` si l'attribut a été spécifié, `false` sinon

Créer et ajouter des éléments

Pour créer un nouvel élément dans un document HTML, utilisez la méthode document.createElement (). Dans ce cas, une balise de début et une balise de fin du type que vous spécifiez seront ajoutées au document.

Le Listing 10.8 montre un exemple d'utilisation de cette méthode pour créer dynamiquement une liste dans un document HTML à partir d'un tableau.

LISTING 10.8 : Utiliser document.createElement () pour générer une liste à partir d'un tableau.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
    <title>Génération d'une liste</title>
  </head>
  <body>
    <h1>Voici quelques types de balles</h1>
    <ul id="ballList">
    </ul>

    <script>
    var typeOfBall = ["basket", "rugby", "tennis", "foot", "hand"];
    for (i=0; i<typeOfBall.length; i++) {
      var listElement = document.createElement("li");
      listElement.innerHTML = typeOfBall[i];
      document.getElementById("ballList").appendChild(listElement);
    }
    </script>

  </body>
</html>
```

Supprimer des éléments

Malgré toutes les grandes choses qu'il vous permet d'accomplir avec vos documents HTML, le modèle DOM n'est pas toujours apprécié des programmeurs JavaScript professionnels. Il a en effet quelques bizarreries et tend à rendre les choses plus difficiles qu'elles ne devraient l'être.

L'un des grands défauts du DOM, c'est qu'il ne fournit aucun moyen permettant de supprimer directement un élément d'un document. Pour y arriver, vous devez demander à DOM de trouver le parent de l'élément que vous voulez supprimer, puis demander à ce parent d'effectuer le sale travail. Cela peut paraître un peu confus, mais le Listing 10.9 devrait rendre cela plus clair.

LISTING 10.9 : Supprimer un élément d'un document.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
    <title>Suppression d'un élément</title>
    <script>
      function removeFirstParagraph(){
        var firstPara = document.getElementById("firstparagraph");
         firstPara.parentNode.removeChild(firstPara);
      }
    </script>
  </head>
  <body>
    <div id="charabia">
      <p id="firstparagraph">Lorem ipsum dolor sit amet, consectetur
adipiscing
elit. Vestibulum molestie pulvinar ante, a volutpat est sodales et. Ut
gravida justo ac leo euismod, et tempus magna posuere. Cum sociis natoque
penatibus et magnis dis parturient montes, nascetur ridiculus mus.
Integer non mi
 iaculis, facilisis risus et, vestibulum lorem. Sed quam ex, placerat nec
tristique id, mattis fringilla ligula. Maecenas a pretium justo.
Suspendisse sit
amet nibh consectetur, tristique tellus quis, congue arcu. Etiam
pellentesque
dictum elit eget semper. Phasellus orci neque, semper ac tortor ac,
laoreet
ultricies enim.</p>
    </div>
    <button onclick="removeFirstParagraph();">On efface tout !</button>
  </body>
</html>
```

Lorsque vous exécutez le code du Listing 10.9 dans un navigateur et que vous cliquez sur le bouton, l'événement `onclick` appelle la fonction `removeFirstParagraph ()`.

La première chose que fait la fonction `removeFirstParagraph ()`, c'est de sélectionner l'élément que nous voulons supprimer, donc celui dont l'ID est `firstparagraph`. Le script sélectionne ensuite le nœud parent du premier paragraphe. Arrivé là, il utilise la méthode `removeChild ()` pour se débarrasser de l'enfant de trop, en l'occurrence justement ce premier paragraphe.

Chapitre 11

Utiliser des événements dans JavaScript

DANS CE CHAPITRE :

- **Découvrir ce qui se passe**
- **Utiliser des gestionnaires pour répondre aux événements**
- **Connaître les types de gestionnaires d'événements**

« Et maintenant, la séquence des événements dans un ordre quelconque. »

Dan Rather

Les pages Web sont bien plus que juste des choses affichant du texte et des images de manière statique. JavaScript apporte aux pages Web de l'interactivité et la possibilité d'effectuer des tâches utiles. Et une partie importante de cette capacité réside dans le fait qu'il est capable de répondre aux événements.



N'oubliez pas de visiter le site Web compagnon de ce livre pour y télécharger les exemples de ce chapitre !

Connaître vos événements

Les *événements* sont des choses qui se produisent dans le navigateur (par exemple, le chargement d'une page), ou encore des choses que fait l'utilisateur (comme cliquer, appuyer sur une touche du clavier, déplacer la souris, et ainsi de suite). Dans le navigateur, des événements se produisent tout le temps.

Tableau 11.1 : Événements pris en charge par tous les éléments HTML.

Événement	Se produit quand…
`abort`	Le chargement d'un fichier est avorté
`change`	Une valeur d'un élément a changé depuis que le focus a été perdu ou récupéré
`click`	Un clic s'est produit sur un élément
`dbclick`	Un double-clic s'est produit sur un élément
`input`	La valeur d'un élément <input> ou <textarea> est modifiée
`keydown`	Une touche est pressée
`keyup`	Une touche est relâchée après avoir été pressée
`mousedown`	Un bouton de la souris est pressé alors que le pointeur se trouve au-dessus d'un élément

`mouseenter`	Le pointeur de la souris est déplacé sur l'élément auquel est attaché le gestionnaire d'événement
`mouseleave`	Le pointeur de la souris quitte l'élément auquel est attaché le gestionnaire d'événement
`mousemove`	Le pointeur de la souris est déplacé au-dessus d'un élément
`mouseout`	Le pointeur de la souris quitte l'élément auquel est attaché le gestionnaire d'événement ou un de ses enfants
`mouseover`	Le pointeur de la souris est déplacé sur l'élément auquel est attaché le gestionnaire d'événement ou un de ses enfants
`mouseup`	Un bouton de la souris est relâché alors que le pointeur se trouve au-dessus d'un élément
`mousewheel`	La molette de la souris a été actionnée
`onreset`	Un formulaire a été réinitialisé
`select`	Du texte a été sélectionné
`submit`	Un formulaire est soumis

Le modèle HTML DOM fournit à JavaScript la possibilité d'identifier les événements qui se produisent dans le navigateur Web et d'y répondre. Les événements peuvent être divisés en plusieurs groupes selon les éléments HTML ou les objets de navigation auxquels ils s'appliquent. La Tableau 11.1 liste les événements qui sont pris en charge par chaque élément HTML.

D'autres types d'événements sont pris en charge par tous les éléments HTML *sauf* `body` et `frameset`. Ils sont décrits dans le Tableau 11.2.

Tableau 11.2 : Événements pris en charge par chaque élément, à l'exception de <body> et <frameset>.

Événement	Se produit quand…
`blur`	Un élément a perdu le focus
`error`	Un fichier n'a pas pu être chargé
`focus`	Un élément a reçu le focus
`load`	Le chargement d'un fichier, ainsi que de ses fichiers attachés, est terminé
`resize`	Le document a été redimensionné
`scroll`	Un défilement a été appliqué au document ou à un élément
`afterprint`	L'aperçu avant impression du document a été refermé, ou le document a commencé à être imprimé
beforeprint	L'aperçu avant impression du document est ouvert, ou le document est prêt à être imprimé
`beforeunload`	La fenêtre, le document et ses fichiers inclus sont sur le point d'être déchargés

`hashchange`	La partie de l'URL placée après le signe dièse (#) change
`pagehide`	Le navigateur place une page dans l'historique de navigation
`pageshow`	Le navigateur accède à une page de l'historique de session
`popstate`	L'élément actif de l'historique de la session change
`unload`	Le document ou un fichier inclus est sur le point d'être déchargé

En plus de ces événements, il en existe en fait des quantités d'autres. Par exemple, l'API `File` contient des séries d'événements dédiés au chargement des fichiers, et la spécification Media de HTML5 gère des événements pour la diffusion de contenus audio et vidéo. En fait, des tas de choses peuvent se produire dans votre navigateur !

Pour une liste complète de tous ces événements, vous pouvez visiter le site `https://developer.mozilla.org/fr/docs/Web/Events`.

Gérer les événements

Lorsque JavaScript réagit en réponse à un certain événement, on parle de *gestion d'événement* (et donc aussi de *gestionnaire d'événement*).

Au fil des ans, les navigateurs se sont vus implémenter de multiples manières pour les programmes JavaScript de gérer les événements. Du même coup, les questions d'incompatibilité entre navigateurs se sont également accrues.

De nos jours, JavaScript en est arrivé au stade où les anciennes et inefficaces méthodes de gestion des événements sont arrivées pratiquement en fin de vie. Cependant, ces vieilles techniques sont encore largement employées, et il est donc important de se pencher dessus.

Utiliser des gestionnaires d'événements inline

Le premier système de gestion des événements date des débuts de JavaScript. Il s'appuie sur des attributs spécifiques, dont notamment le gestionnaire d'événement `onclick`.

Les attributs de gestion d'événement dits *inline* sont formés en ajoutant le préfixe `on` à un événement. Pour les utiliser, ajoutez l'attribut d'événement à un élément HTML. Lorsque l'événement spécifié se produit, le code JavaScript contenu dans la valeur de l'attribut est exécuté. Par exemple, le Listing 11.1 affiche un message d'alerte lorsque l'utilisateur clique sur le lien.

LISTING 11.1 : Affichage d'un message d'alerte.

```
<a href="home.html" onclick="alert('Retour à l'accueil !');">
Cliquez ici pour retourner à l'accueil</a>
```

Si vous insérez cette balise dans un document HTML et que vous cliquez sur le lien, vous verrez apparaître une fenêtre de message avec les mots *Retour à l'accueil !* Si vous cliquez sur le bouton OK de la fenêtre, le lien traite le gestionnaire d'événement associé à l'élément `a`, autrement dit ici l'activation du lien spécifié dans l'attribut `href`, soit la page `home.html`.

Dans de nombreuses situations, vous ne voulez pas que l'action associée par défaut à un élément soit exécutée. Par exemple, vous pourriez préférer, dans le cas du Listing 11.1, que seule la fenêtre de message soit affichée, sans qu'il se produise autre chose.

Les programmeurs JavaScript ont développé plusieurs techniques différentes pour contourner ce genre de difficulté. Une méthode consiste à faire de l'action par défaut quelque chose de neutre, sans conséquence. Dans notre exemple, il suffirait de changer la valeur de l'attribut `href` en #, et le lien pointera alors vers lui-même :

```
<a href="#" onclick="alert('Retour à l'accueil !');">
Cliquez ici pour retourner à l'accueil</a>
```

Une meilleure méthode, cependant, consiste à demander au gestionnaire d'événement de renvoyer une valeur booléenne `false`, afin d'indiquer que l'action par défaut ne doit pas être exécutée :

```
<a href="home.html" onclick="alert('Retour à l'accueil !');return false">
Cliquez ici pour retourner à l'accueil</a>
```

Gérer les événements en utilisant les propriétés des éléments

L'un des plus grands problèmes avec les anciennes techniques d'affectation d'événements à des éléments est que cela viole une des meilleures techniques de programmation : séparer la présentation (ce à quoi quelque chose ressemble) de la fonctionnalité (ce que fait l'élément). Mixer les gestionnaires d'événements et les balises HTML rend vos pages Web plus difficiles à gérer, à déboguer et à comprendre.

À partir de la version 3 de son navigateur, Netscape a introduit un nouveau modèle d'événement qui permet aux programmeurs d'attacher des événements à des éléments sous la forme de propriétés. Le Listing 11.2 illustre un exemple de fonctionnement de ce modèle.

LISTING 11.2 : Attacher des événements à des éléments en utilisant des propriétés.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
    <title>Comptage</title>
    <script>
    // attend que la fenêtre soit finie de charger avant d'enregistrer
les événements
    window.onload = initializer;

    // crée une variable globale pour le comptage
    var theCount = 0;

    /**
    Enregistre l'événement onclick
    */
    function initializer(){
    document.getElementById("incrementButton").onclick = increaseCount;
    }

    /**
    Incrémente theCount et affiche le résultat.
    */
    function increaseCount(){
      theCount++;
   document.getElementById("currentCount").innerHTML = theCount;
  }

  </script>
</head>
<body>
  <h1>Cliquez sur le bouton pour compter.</h1>
   <p>Valeur courante : <span id="currentCount">0</span></p>
  <button id="incrementButton">Incrémenter le compteur</button>
</body>
</html>
```

L'un des points à noter dans le Listing 11.2 est le fait que les noms de fonctions qui sont affectés au gestionnaire d'événement ne comportent pas de parenthèses. Cela signifie que l'ensemble de la fonction est assigné au gestionnaire d'événement, ce qui revient à lui indiquer qu'il doit s'exécuter lorsque cet événement se produit, au lieu d'utiliser pour cela un appel de fonction. Si vous ajoutez des parenthèses après le nom de la fonction, celle-ci sera exécutée, et son résultat sera transmis à l'événement `onclick`, ce qui n'est pas ce que nous voulons obtenir ici.

Ajouter un gestionnaire d'événement en utilisant addEventListener

Même si les deux méthodes précédentes sont utilisées très couramment, et sont prises en charge par n'importe quel navigateur, une méthode plus moderne et plus souple de gérer les événements consiste à faire appel à la méthode `addEventListener ()` (et c'est bien sûr celle que nous recommandons avec les navigateurs actuels).

La méthode `addEventListener ()` surveille les événements de n'importe quel nœud DOM et déclenche les actions basées sur ces événements. Lorsque la fonction spécifiée en tant qu'action pour l'événement est exécutée, elle reçoit automatiquement un argument unique, l'objet `Event`. Par convention, nous appelons cet argument `e`.

La méthode `addEventListener ()` offre plusieurs avantages sur les attributs d'événement de DOM :

- Vous pouvez appliquer plusieurs « écouteurs » d'événements à un même élément.
- Cette méthode fonctionne avec n'importe nœud de l'arborescence DOM, pas simplement sur des éléments.
- Elle vous procure un contrôle accru sur les événements.

Le Listing 11.3 illustre l'emploi de la méthode `addEventListener ()`. Cet exemple contient la même fonction de comptage que celle du Listing 11.2, mais elle lui ajoute un second gestionnaire d'événement qui augmente la taille du nombre chaque fois que l'utilisateur clique sur le bouton.

LISTING 11.3 : Définir un événement avec addEventListener ().

```
<html>
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
    <title>Comptage</title>
    <script>
    // attend que la fenêtre soit finie de charger avant d'enregistrer
les événements
    window.addEventListener('load',registerEvents,false);

    // crée une variable globale pour le comptage
    var theCount = 0;

    /**
    Enregistre les événements onclick
    */
    function registerEvents(e){
    document.getElementById("incrementButton").addEventListener('click',
        increaseCount,false);
    document.getElementById("incrementButton").addEventListener('click',
        changeSize,false);

    }
```

```
  /**
  Incrémente theCount et affiche le résultat.
  */
  function increaseCount(e){
    theCount++;
    document.getElementById("currentCount").innerHTML = theCount;
  }
  /**
  Change la taille des caractères pour le compteur
  */
  function changeSize(e){
    document.getElementById("currentCount").style.fontSize = theCount;
  }
  </script>
</head>
<body>
  <h1>Cliquez sur le bouton pour compter.</h1>
  <p>Valeur courante : <span id="currentCount">0</span></p>
  <button id="incrementButton">Incrémenter le compteur</button>
</body>
</html>
```

La Figure 11.1 illustre la sortie produite par le Listing 11.3 (d'accord, nous avons un peu triché en faisant débuter le compteur à 300 dans le fichier téléchargeable).



FIGURE 11.1 : Attacher deux événements au même élément multiplie les possibilités !

La méthode `addEventListener ()` est implémentée en utilisant trois arguments.

Le premier argument est le type de la fonction. Contrairement aux deux autres méthodes de gestion des événements, `addEventListener ()` demande simplement le nom de l'événement, sans le préfixe `on`.

Le second argument est la fonction à appeler lorsque l'événement est déclenché. Comme dans le cas de la méthode de gestion des événements *via* les propriétés, il est important de ne pas utiliser les parenthèses à la suite du nom de la fonction. En effet, vous voulez passer celle-ci pour qu'elle soit exécutée par le gestionnaire, et non le résultat qu'elle renverrait sinon immédiatement.

Le troisième argument est une valeur booléenne (`true` ou `false`) qui indique l'ordre dans lequel les gestionnaires d'événements sont exécutés lorsqu'un élément associé à un événement possède un élément parent qui est, lui aussi, associé à un événement.

Lorsque plusieurs éléments sont imbriqués, il est indispensable de savoir lequel doit être traité en premier. Par exemple, la Figure 11.2 illustre un problème courant. Le carré extérieur est cliquable, mais le cercle intérieur également. Si vous cliquez sur le cercle intérieur, quel est l'événement qui doit être déclenché en premier : celui qui est attaché au carré, ou celui qui est attaché au cercle ?

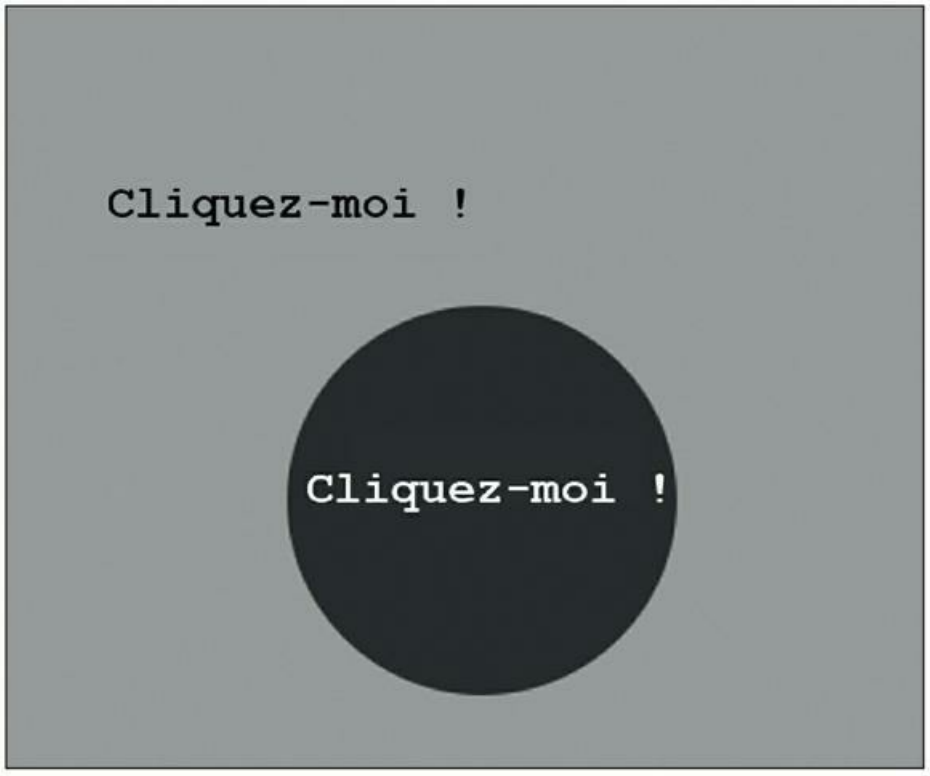


FIGURE 11.2 : Des événements à l'intérieur d'autres événements.

Pour la plupart des gens, la logique voudrait que l'événement lié au cercle intérieur soit traité en premier. Cependant, lorsque Microsoft implémenta sa propre version des événements dans Internet Explorer, la décision prise fut de donner la priorité à l'événement extérieur (le carré).

La manière la plus courante de gérer les événements dans ce genre de circonstance est appelée par les programmeurs *bubbling up* (disons un bouillonnement). Les événements de l'élément le plus intérieur sont déclenchés en premier, puis les « bulles » remontent jusqu'aux éléments extérieurs. Pour utiliser cette méthode, le dernier argument de `addEventListener ()` doit être défini comme étant `false`, ce qui est aussi la valeur par défaut.

L'autre manière de gérer ce scénario est appelée la méthode par *capture*. Dans le mode capture, les éléments les plus extérieurs sont prioritaires, ceux qui sont le plus à l'intérieur étant traités en dernier. Dans ce cas, le dernier argument de `addEventListener ()` prend la valeur `true`.

Le Listing 11.4 montre combien le fait de connaître l'ordre dans lequel les gestionnaires d'événements sont exécutés est important. Les éléments `h1` sont associés à des événements `click`, mais les mots contenus dans ces en-têtes aussi.

LISTING 11.4 : Événements par capture et événements par bouillonnement.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
  <title>Événements : Capture vs. Bouillonnement</title>
  <style>
    #theText {font-size: 18px;}
    h1 {
      border:1px solid #000;
      background-color: #dadada;
    }
```

```
    #capEvent, #bubEvent {
      background-color: #666;
    }
  </style>

  <script>
  // attend que la fenêtre soit finie de charger avant d'enregistrer les
événements
  window.addEventListener(‘load',registerEvents,false);

  /**
  Enregistre les événements onclick
  */
  function registerEvents(e){

document.getElementById("capTitle").addEventListener(‘click',makeTiny,true);


document.getElementById("capEvent").addEventListener(‘click',makeHuge,true);


document.getElementById("bubTitle").addEventListener(‘click',makeTiny,false)


document.getElementById("bubEvent").addEventListener(‘click',makeHuge,false)

  }

  function makeHuge(e){
    console.log("agrandissement du texte");
    document.getElementById("theText").style.fontSize = "80px";
  }
  function makeTiny(e){
    console.log("réduction du texte");
    document.getElementById("theText").style.fontSize = "10px";
  }
  </script>
</head>
<body>
  <h1 id="capTitle">Événement <span id="capEvent"> (mode Capture)</span>
</h1>

  <h1 id="bubTitle">Événement <span id="bubEvent">(mode Bouillonnement)
</span></h1>
  <p id="theText">Bonjour, les Événements!</p>
</body>
```

```
</html>
```

La Figure 11.3 illustre le comportement de ce programme dans un navigateur.

Sur la Figure 11.3, le premier clic a eu lieu sur les mots « (mode Capture) ». Dans ce cas, l'événement associé au conteneur le plus grand est déclenché en premier, suivi par l'événement associé au texte « (mode Capture) ».

Si vous cliquez sur « (mode Bouillonnement) », par contre, c'est l'événement qui lui est associé qui est déclenché en premier, suivi de l'événement associé à son élément parent.

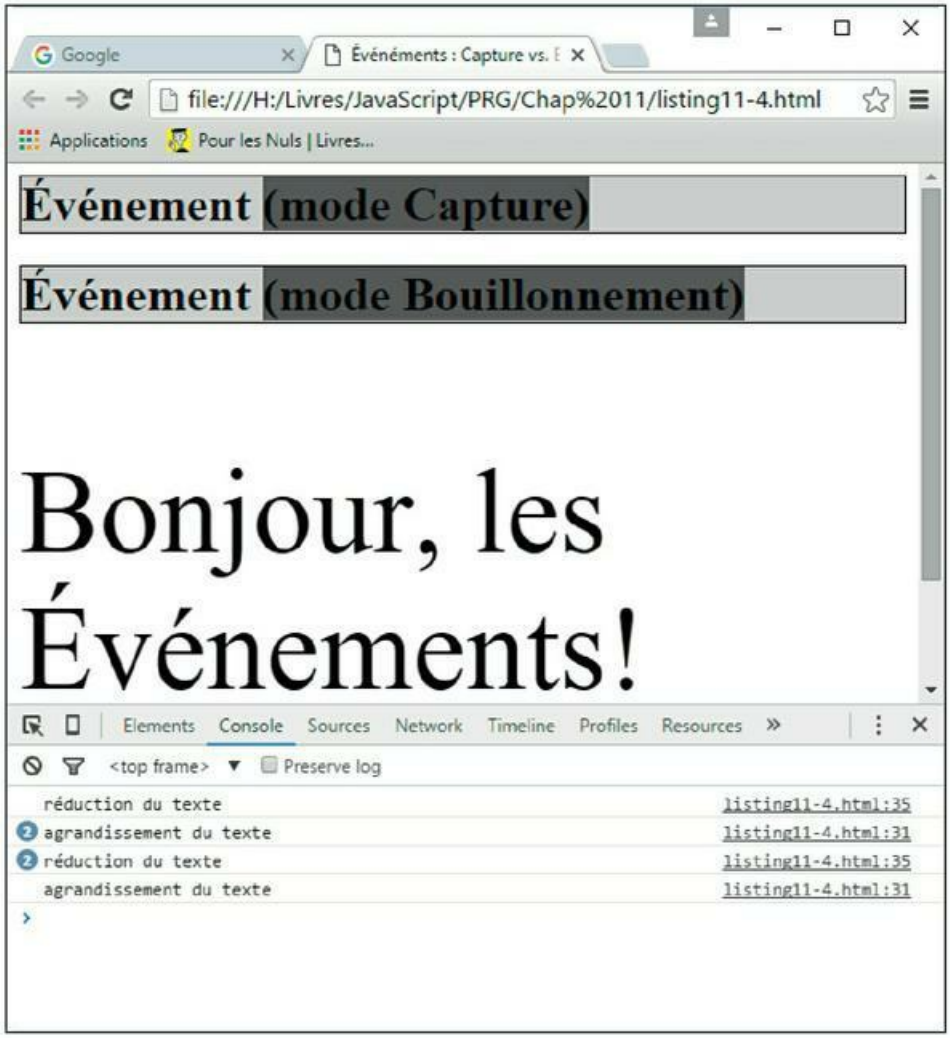


FIGURE 11.3 : Gérer des événements imbriqués.

Stopper la propagation

Il existe en fait une troisième possibilité pour gérer des événements imbriqués : vous déclenchez le premier événement et vous arrêtez tout de suite. Vous pouvez désactiver la propagation d'un événement, et même de tous les événements, en utilisant la méthode `stopPropagation ()`.



Si vous n'avez pas besoin de propager les événements dans votre script, il est conseillé de désactiver les deux premières méthodes, car elles consomment des ressources système et peuvent rendre votre site Web plus lent.

Le Listing 11.5 illustre l'utilisation de cette troisième méthode.

LISTING 11.5 : Désactiver la propagation des événements.

```
function load(e){

  if (!e) var e = window.event;

  //configure cancelBubble pour IE 8 et avant
  e.cancelBubble = true;

  if (e.stopPropagation) e.stopPropagation();


document.getElementById("capTitle").addEventListener(‘click',makeTiny,true);


document.getElementById("capEvent").addEventListener(‘click',makeHuge,true);


document.getElementById("bubTitle").addEventListener(‘click',makeTiny,false)


document.getElementById("bubEvent").addEventListener(‘click',makeHuge,false)

}
```

Chapitre 12

Entrer et sortir

DANS CE CHAPITRE :

- **Travailler avec les formulaires**
- **Accepter des entrées**
- **Envoyer des sorties**

« Dysfonctionnement. Nécessite entrée. »

Number 5, *Short Circuit* (1986)

Gérer les entrées de l'utilisateur et renvoyer des résultats sont des fonctions de base et évidemment nécessaires pour n'importe quel programme d'ordinateur. Dans ce chapitre, vous allez découvrir comment JavaScript et HTML travaillent de concert pour recevoir et envoyer des données.



N'oubliez pas de visiter le site Web compagnon de ce livre pour y télécharger les exemples de ce chapitre !

Comprendre les formulaires HTML

La première manière permettant d'obtenir des saisies utilisateur dans des applications Web consiste à faire appel aux formulaires HTML. Les formulaires HTML offrent aux développeurs la possibilité de créer des champs de texte, des listes déroulantes, des boutons radio, des cases à cocher et des boutons. Avec CSS, vous pouvez ajuster l'aspect d'un formulaire en fonction du style de votre site Web. JavaScript vous permet en plus d'améliorer les fonctionnalités de vos formulaires.

L'élément form

Les formulaires HTML sont contenus dans un élément `form`. Celui-ci est le conteneur dans lequel prennent place les champs de saisie, les boutons, les cases à cocher et les étiquettes qui définissent une zone de saisie utilisateur. L'élément `form` fonctionne pour l'essentiel à l'instar de tout autre conteneur, comme `div`, `article` ou `section`. Mais il contient également certains attributs qui indiquent au navigateur ce qu'il doit faire avec les champs de saisie qu'il contient.

Le Listing 12.1 illustre cela en proposant un formulaire HTML contenant deux champs de saisie et un bouton de soumission.

LISTING 12.1 : Exemple de page HTML contenant un formulaire.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
    <title>Formulaire HTML</title>
    <style>
      form {margin-top: 20px;}
    </style>
  </head>
  <body>

    <form action="subscribe.php" name="newsletterSubscribe"
method="post">
    <label for="firstName">Prénom : </label>
    <input type="text" name="firstName" id="firstName"><br><br>
    <label for="email">Email : <input type="text" name="email"
id="email"></label>
       <br><br>
    <input type="submit" value="Abonnez-vous à notre newsletter!">
    </form>

  </body>
</html>
```

La Figure 12.1 illustre l'exécution de ce code dans un navigateur.

Dans l'exemple précédent, l'élément `form` possède trois attributs :

- `action` : Indique au navigateur ce qu'il doit faire avec la saisie de l'utilisateur. En règle générale, cette action demande l'exécution d'un script côté serveur.
- `name` : Spécifie le nom que le programmeur a affecté à ce formulaire. Cet attribut est notamment utile pour accéder au formulaire en utilisant le modèle DOM.
- `method` : Prend comme valeur `get` ou `post`, ce qui indique si le navigateur devrait envoyer les données du formulaire ver l'URL ou vers l'en-tête HTML.

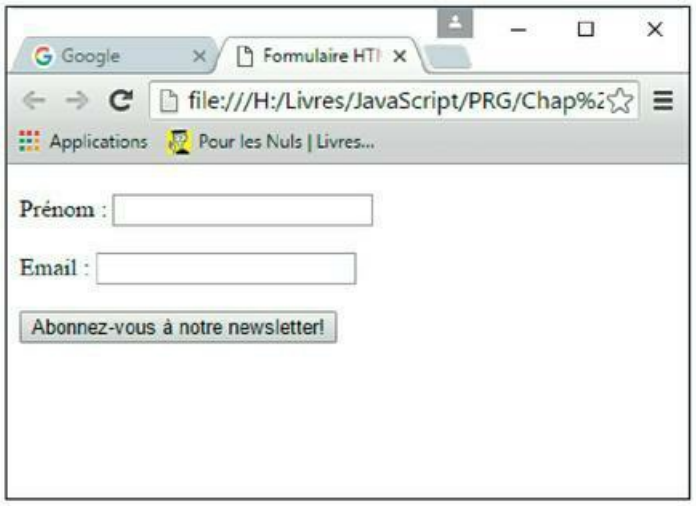


FIGURE 12.1 : Un formulaire HTML.

L'élément `form` peut également contenir plusieurs autres attributs :

- `accept-charset` : Indique le jeu de caractères accepté par le serveur. À moins de travailler avec un contenu multilingue (et même dans ce cas), vous pouvez laisser sans danger cet attribut de côté.
- `autocomplete` : Indique si les champs de saisie du formulaire doivent utiliser la fonction de saisie automatique du navigateur.
- `enctype` : Indique le type de contenu que le formulaire devrait soumettre au serveur. Dans le cas de formulaires qui ne traitent que du texte, cet attribut devrait prendre la valeur `text/html`. Si votre formulaire doit soumettre un fichier au serveur (par exemple pour télécharger une image), la valeur de cet attribut devrait être `multipart/form-data`. La valeur par défaut est `application/x-www-form-urlencoded`.
- `novalidate` : Valeur booléenne indiquant si ce qui est saisi dans le formulaire devrait être validé par le navigateur lors de la soumission. Si cet attribut n'est pas spécifié, les formulaires sont validés par défaut.
- `target` : Indique si la réponse renvoyée par le serveur devrait être affichée après soumission du formulaire. La valeur par défaut (`"_self"`) demande à ouvrir la réponse dans la même fenêtre de navigation que celle qui contient le formulaire. Une autre option consiste à afficher cette réponse dans une nouvelle fenêtre (`"_blank"`). Il est également possible de spécifier un cadre nommé.

L'élément label

Vous pouvez utiliser l'élément `label` pour associer une description (ou étiquette) à un champ de saisie. L'attribut `for` de cet élément prend la valeur de l'attribut `id` de l'élément auquel l'étiquette devrait être associée, comme dans l'exemple suivant :

```
<label for="firstName">Prénom : </label>
<input type="text" name="firstName">
```

Une autre méthode consiste à imbriquer le champ du formulaire à l'intérieur de l'élément `label`, comme ceci :

```
<label>Prénom : <input type="text" name="firstName"></label>
```

L'avantage est ici de ne pas avoir à attribuer un identificateur au champ de saisie (ce qui revient bien souvent à dupliquer son attribut `name`).

L'élément input

L'élément HTML `input` est le plus fondamental en ce qui concerne les formulaires. Selon la valeur de son attribut `type`, il fait afficher (ou non) par le navigateur plusieurs types de champs de saisie.

Le plus souvent, le type de l'élément `input` est défini avec la valeur `"text"`, ce qui demande au navigateur de créer un champ pour la saisie de texte. L'attribut facultatif `value` sert à affecter une valeur par défaut à l'élément, et l'attribut `name` est le nom qui est associé à la valeur pour former un couple nom/valeur qui est accessible *via* le modèle DOM, et qui est transmis avec le reste des valeurs du formulaire lorsque celui-ci est soumis.

Un champ de saisie basique de type texte se présente donc de la manière suivante :

```
<input type="text" name="nomVille">
```

Avec HTML5, l'élément `input` s'est vu doter d'un tas de nouvelles valeurs d'attribut possibles pour `type`. Ces nouvelles valeurs permettent au développeur Web de spécifier plus précisément la nature des données qui devraient être saisies dans le champ du formulaire. Elles permettent également au navigateur de fournir des contrôles mieux adaptés aux différents types de données, et par voie de conséquence de créer de meilleures applications Web.



Il pourrait sembler un peu étrange que ce chapitre se concentre autant sur les fonctionnalités de gestion des formulaires de HTML, plutôt que de s'attaquer tout de suite à JavaScript. Cependant, les formulaires constituent un domaine dans lequel HTML peut réellement réduire la charge de travail des programmeurs, et il est donc vital que les développeurs JavaScript soient au clair sur ce sujet.

Les valeurs possibles pour l'attribut `type` de l'élément `input` sont décrites dans le Tableau 12.1.

Tableau 12.1 : Valeurs possibles pour l'attribut type de l'élément input.

Valeur	Description
`button`	Un bouton cliquable
`checkbox`	Une case à cocher
`color`	Un sélecteur de couleur
`date`	Un contrôle de date (année, mois et jour)
`datetime`	Un contrôle de date et d'heure (année, mois, jour, heure, minute, seconde et fraction de seconde en temps universel coordonné – TUC ou UTC)
`datetime-local`	Un contrôle de date et d'heure (année, mois, jour, heure, minute, seconde et fraction de seconde sans précision de zone)
`email`	Un champ pour une adresse de messagerie
`file`	Un champ de sélection de fichier avec un bouton Parcourir

`hidden`	Un champ de saisie caché
`image`	Un bouton Soumettre utilisant une image à la place du nom par défaut
`month`	Un contrôle mois et année
`number`	Un champ de saisie numérique
`password`	Un champ de saisie pour un mot de passe
`radio`	Un bouton radio
`range`	Un champ de saisie utilisant une plage de nombres, comme un contrôle avec glissière
`reset`	Un bouton Réinitialiser
`search`	Un champ de texte pour saisir un critère de recherche
`submit`	Un bouton Soumettre
`tel`	Un champ pour entrer un numéro de téléphone
`text`	Un champ pour saisir une unique ligne de texte (c'est la valeur par défaut)
`time`	Un contrôle pour entrer une heure (pas de zone de temps)
`url`	Un champ pour saisir une URL
`week`	Un contrôle semaine et année (pas de zone de temps)



Au moment où ce livre est écrit, tous les navigateurs ne prennent pas en charge l'ensemble des valeurs possibles pour l'attribut `type` de l'élément `input`. Dans ce cas, le navigateur se contentera d'afficher le type de champ par défaut, c'est-à-dire `text`.

L'élément select

L'élément HTML `select` définit soit une liste déroulante, soit une saisie avec sélection multiple. Cet élément contient une série d'options, qui sont les choix proposés à l'utilisateur dans le contrôle `select`, comme l'illustre le Listing 12.2.

LISTING 12.2 : Une liste déroulante créée en utilisant l'élément HTML select.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
    <title>Formulaire HTML</title>
    <style>
      form {margin-top: 20px;}
    </style>
  </head>
  <body>

    <form action="subscribe.php" name="newsletterSubscribe"
method="post">
      <select name="favoriteColor">
        <option value="red">rouge</option>
        <option value="blue">bleu</option>
        <option value="green">vert<option>
      </select>
    </form>

  </body>
</html>
```

La Figure 12.2 montre le code du Listing 12.2 en action.

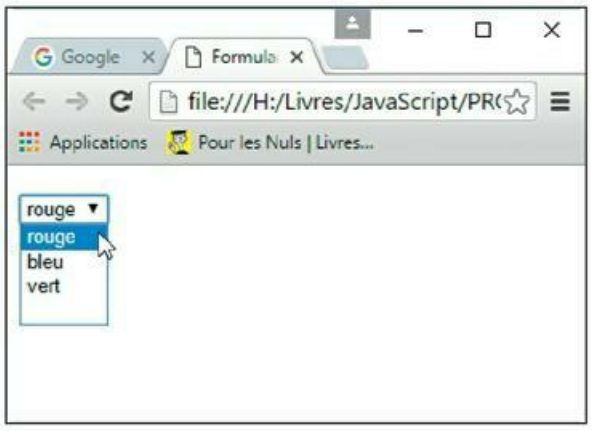


FIGURE 12.2 : Une liste déroulante créée en HTML.

L'élément textarea

L'élément `textarea` définit un champ de saisie multiligne. Dans l'exemple qui suit, ce champ comporte quatre lignes sur une largeur de trente caractères :

```
<textarea name="description" rows="4" cols="30"></textarea>
```

L'élément button

L'élément `button` fournit une autre manière de créer un bouton cliquable :

```
<button name="monBouton">Cliquez-moi ! </button>
```

L'élément `button` peut être utilisé à la place d'éléments `input` dans lesquels l'attribut `type` est défini avec la valeur `"submit"`. Mais vous pouvez également vous en servir partout où vous avez besoin d'un bouton, sans pour autant vouloir soumettre quoi que ce soit.



Si vous ne voulez pas que le bouton soumette le formulaire lorsque l'utilisateur clique dessus, vous devez lui ajouter un attribut `type` ayant pour valeur `"button"`.

Travailler avec l'objet Form

Le modèle HTML DOM représente les formulaires en utilisant l'objet `Form`. *Via* cet objet, vous pouvez récupérer et définir la valeur des champs d'un formulaire, contrôler l'action à exécuter lorsque l'utilisateur soumet un formulaire, et modifier le comportement de celui-ci.

Utiliser les propriétés de l'objet Form

Les propriétés de l'objet `Form` correspondent aux attributs de l'élément HTML `form` (reportez-vous à la première partie de ce chapitre). Ils sont utilisés pour retrouver ou définir les valeurs de l'élément HTML `form` avec JavaScript. Le Tableau 12.2 liste toutes les propriétés de l'objet `Form`.



Les objets DOM sont des représentations des pages HTML. Leur but est de vous donner accès à différentes parties du document *via* JavaScript (ce que l'on appelle l'*interface de programmation*). Tout ce qui se trouve dans un document HTML peut être récupéré et modifié en utilisant le modèle DOM.

Tableau 12.2 : Les propriétés de l'objet Form.

Propriété	Utilisation
`acceptCharset`	Récupère ou définit une liste des jeux de caractères pris en charge par le serveur
`action`	Lit ou définit la valeur de l'attribut `action` de l'élément `form`
`autocomplete`	Lit ou définit l'état de la fonction de saisie automatique du navigateur
`encoding`	Indique au navigateur comment encoder les données du formulaire (comme texte ou en tant que fichier), synonyme de enctype
`enctype`	Indique au navigateur comment encoder les données du formulaire (comme texte ou en tant que fichier)
`length`	Récupère le nombre de contrôles dans le formulaire
`method`	Récupère ou définit la méthode HTTP que le navigateur utilise pour soumettre le formulaire.
`name`	Récupère ou définit le nom du formulaire

`noValidate`	Indique que le formulaire n'a pas besoin d'être validé avant soumission
`target`	Indique où doivent être affichés les résultats du formulaire qui a été soumis

UTILISER L'ATTRIBUT AUTOCOMPLETE

L'attribut `autocomplete` d'un élément HTML `form` définit la valeur par défaut de la saisie automatique pour les éléments `input` d'un formulaire. Si vous voulez que le navigateur propose de compléter automatiquement la saisie pour chaque champ de saisie du formulaire, affectez la valeur `'on'` à l'attribut `autocomplete`.

Si vous voulez être capable de sélectionner les éléments pour lesquels la saisie automatique est activée, ou si votre document gère lui-même cette fonctionnalité (*via* JavaScript), donnez à l'attribut `autocomplete` du formulaire la valeur `'off'`. Vous pourrez alors personnaliser cet attribut pour chaque élément `input` individuel du formulaire.

Le Chapitre 10 vous propose des techniques permettant de récupérer ou de définir la valeur des propriétés d'un formulaire. Après avoir référencé un formulaire en utilisant une de ces méthodes, vous pouvez ensuite accéder à la propriété voulue en utilisant la notation point ou des crochets droits.

Pour obtenir par exemple la propriété `name` du premier formulaire du document, vous pourriez écrire l'instruction suivante :

```
document.getElementByTagName ("form")[0].name
```

Une manière plus courante d'accéder à un formulaire consiste à lui assigner un attribut `id`, puis à utiliser `getElementById` pour le sélectionner.

Le modèle DOM fournit une autre méthode, plus pratique, pour accéder aux formulaires : la *collection* `forms`. Avec cette collection, vous pouvez accéder aux formulaires d'un document de deux manières :

- **Par un numéro d'indice :** Lorsqu'un élément `form` est créé dans le document, un indice débutant à zéro lui est affecté. Pour accéder à ce formulaire, utilisez la syntaxe `document.forms[0]`.
- **Par le nom :** Vous pouvez aussi accéder aux formulaires en utilisant l'attribut `name` de l'élément `form`. Par exemple, pour obtenir la valeur de la propriété `action` d'un formulaire ayant pour nom `"formSouscription"`, vous pourriez écrire `document.forms.formSouscription.action`. Une autre manière de procéder consisterait à utiliser des crochets droits comme ceci : `document.forms["formSouscription"].action`.

Utiliser les méthodes de l'objet Form

L'objet `Form` possède deux méthodes : `reset ()` et `submit ()`.

La méthode reset ()

La méthode `reset ()` réinitialise toutes les modifications dans les champs du formulaire qui ont été effectuées une fois la page chargée. Ces champs sont donc remis à leur valeur par défaut. Le résultat est identique à celui du bouton HTML Annuler, qui est créé en utilisant un attribut `type="reset"` avec un élément `input`, comme dans le code suivant :

```
<input type="reset" value="Effacer le formulaire">
```

La méthode submit ()

La méthode `submit ()` provoque la soumission des valeurs du formulaire en fonction des propriétés définies pour celui-ci (`action`, `method`, `target`, et ainsi de suite). Le résultat est le même qu'avec le bouton HTML Soumettre, qui est créé en utilisant un attribut `type="submit"` avec un élément `input`, comme dans le code suivant :

```
<input type="submit" value="Soumettre le formulaire">
```

Le Listing 12.3 illustre l'emploi des méthodes `reset ()` et `submit ()`, ainsi que de plusieurs propriétés de l'objet `Form`.

LISTING 12.3 : Utiliser les propriétés de l'objet Form.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
    <title>Abonnez-vous à notre newsletter !</title>
    <script>
      function setFormDefaults(){
        document.forms.subscribeForm.method = "post";
        document.forms.subscribeForm.target = "_blank";
        document.forms.subscribeForm.action = "http://watzthis.us9.list-
manage.com/
           subscribe/post?u=1e6d8741f7db587af747ec056&amp;id=663906e3ba";

        // enregistre les événements
        document.getElementById('btnSubscribe').addEventListener('click',
           submitForm);
        document.getElementById('btnReset').addEventListener('click',
resetForm);
      }
      function submitForm() {
        document.forms.subscribeForm.submit();
      }
      function resetForm() {
        document.forms.subscribeForm.reset();
      }
    </script>
  </head>
  <body onload="setFormDefaults();">
    <form name="subscribeForm">
      <h2>Inscrivez-vous à notre liste de diffusion</h2>
      <label for="mce-EMAIL">Adresse de messagerie </label>
      <input type="email" value="" name="EMAIL" id="mce-EMAIL">
      <button type="button" id="btnSubscribe">Abonnez-vous !</button>
      <button type="button" id="btnReset">Réinitialiser</button>
    </form>
  </body>
</html>
```

Accéder aux éléments du formulaire

JavaScript offre plusieurs techniques permettant d'accéder aux champs de saisie d'un formulaire et à leur valeur. Cependant, toutes ces techniques ne sont pas nées égales, et il existe des différences d'opinions entre programmeurs JavaScript à ce sujet. Dans ce qui suit, nous allons passer en revue ces techniques en signalant les points positifs et les inconvénients de chacune :

» **Utiliser le numéro d'indice du formulaire et de ses champs de saisie.** Par exemple, pour accéder au premier champ de saisie du premier formulaire, vous pourriez utiliser le code suivant :

```
document.forms[0].elements[0]
```



Cette technique est à éviter, car elle repose sur l'idée que la structure du document et l'ordre des éléments à l'intérieur du formulaire ne changera pas. Dès l'instant où quelqu'un déciderait que le champ `email` devrait être maintenant placé avant le champ `nom`, c'est l'ensemble du script qui serait ruiné.

» **Utiliser le nom du formulaire et le nom du champ de saisie.** Par exemple :

```
document.monForm. Nom
```

Cette façon de procéder est plus facile à lire et à utiliser. Elle est prise en charge par tous les navigateurs (de plus, elle existe pratiquement depuis le début du développement du modèle DOM).

» **Utiliser `getElementById` pour sélectionner le formulaire, et le nom du champ de saisie concerné.** Par exemple :

```
document.getElementById ("monForm"). Nom
```

Cette technique nécessite que vous affectiez un attribut `id` à l'élément `form`. Par exemple, le code ci-dessus rechercherait un champ appelé `Nom` dans un élément `form` construit de cette manière :

```
<form id="monForm" action="monaction.php">
. . .
</form>
```

» **Utiliser une valeur d'attribut `id` unique pour le champ afin d'accéder directement à celui-ci.** Par exemple :

```
document.getElementById ("Nom")
```



Dans ce cas, si vous avez plusieurs formulaires sur votre page, vous devrez vous assurer que chaque champ, où qu'il se trouve, possède un attribut `id` parfaitement unique (en fait, comme les attributs `id` doivent de toute manière être uniques, ce n'est pas véritablement un problème).

Obtenir et définir les valeurs des éléments d'un formulaire

Le modèle DOM vous donne accès aux éléments des formulaires (nom et valeur) en utilisant les propriétés `name` et `value`. Le Listing 12.4 illustre cette technique en proposant une calculatrice toute simple.

LISTING 12.4 : Une application de calcul démontrant l'utilisation des champs de saisie d'un formulaire.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
    <title>Les maths amusantes</title>
    <script>

      function registerEvents() {

document.mathWiz.operate.addEventListener('click',doTheMath,false);
      }

      function doTheMath() {
        var first = parseInt(document.mathWiz.numberOne.value);
        var second = parseInt(document.mathWiz.numberTwo.value);
        var operator = document.mathWiz.operator.value;
        switch (operator){
           case "add":
           var answer = first + second;
           break;
          case "subtract":
           var answer = first - second;
           break;
          case "multiply":
           var answer = first * second;
           break;
           case "divide":
           var answer = first / second;
           break;
        }

        document.mathWiz.theResult.value = answer;
      }
    </script>
  </head>
  <body onload="registerEvents();">
    <form name="mathWiz">
      <label>Premier nombre : <input type="number" name="numberOne">
</label><br><br>
      <label>Second nombre : <input type="number" name="numberTwo">
</label><br><br>
      <label>Opérateur :
        <select name="operator">
```

```
        <option value="add"> + </option>
        <option value="subtract"> - </option>
        <option value="multiply"> * </option>
        <option value="divide"> / </option>
      </select></label>
      <br><br>
      <input type="button" name="operate" value="Faites le calcul !"><br>
<br>
      <label>Résultat : <input type="number" name="theResult"></label>
    </form>
  </body>
</html>
```

Le résultat est illustré sur la Figure 12.3.

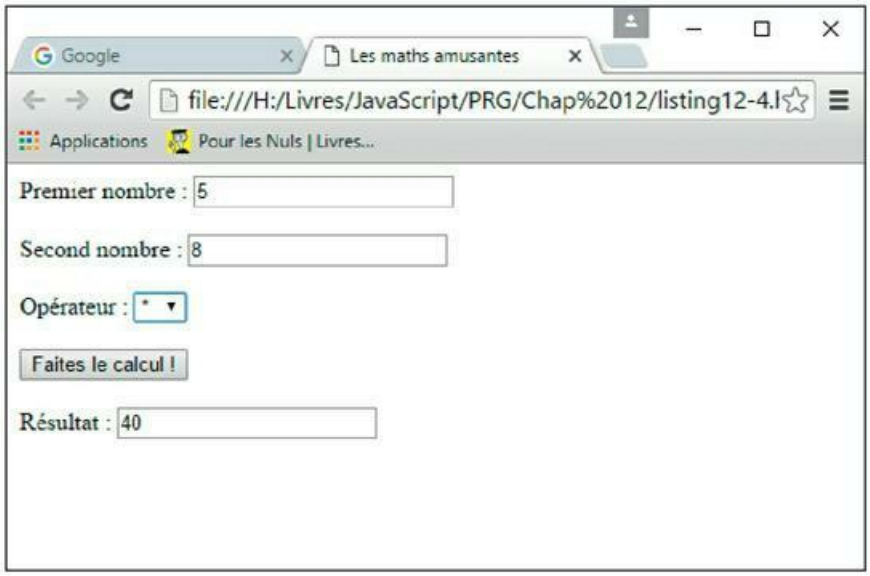


FIGURE 12.3 : Un calculateur de poche.

Valider la saisie de l'utilisateur

L'un des rôles les plus courants de JavaScript consiste à contrôler, ou valider, le contenu du formulaire avant qu'il ne soit soumis au serveur. La validation de formulaire en JavaScript fournit une protection supplémentaire contre des données fausses ou potentiellement dangereuses qui pourraient perturber, voire corrompre, une application Web. Elle permet également de signaler tout de suite à l'utilisateur les erreurs qu'il aurait pu commettre.

Certaines de ces tâches de validation JavaScript ont été remplacées par des attributs HTML depuis l'introduction de HTML5. Cependant, du fait des problèmes de compatibilité HTML5 que connaissent la plupart des navigateurs, il reste préférable de valider les données saisies par l'utilisateur en faisant appel à JavaScript.

Dans le cas de la calculatrice développée sur le Listing 12.4, le type de donnée a été défini comme étant numérique pour les deux opérandes. De cette manière, le navigateur devrait refuser toute autre saisie que des nombres. Pour autant, comme ce type est relativement nouveau, vous ne pouvez pas toujours compter sur le navigateur pour le prendre en charge. C'est pourquoi la validation de la saisie utilisateur par JavaScript reste si importante.

Le Listing 12.5 propose un script de validation de la saisie. La chose importante à noter ici est que l'action du formulaire est configurée pour déclencher la fonction de validation de la saisie. La méthode `submit()` du formulaire n'est exécutée qu'une fois les tests de la fonction de validation exécutés.

LISTING 12.5 : Validation de saisie avec JavaScript.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
    <title>Les maths amusantes</title>
    <script>

      function registerEvents() {

document.mathWiz.operate.addEventListener('click',validate,false);
      }

      function validate() {
        var first = document.mathWiz.numberOne.value;
        var second = document.mathWiz.numberTwo.value;
        var operator = document.mathWiz.operator.value;

        if (/^\d+$/.test(first) && /^\d+$/.test(second)) {

          doTheMath();

        } else {

          alert("Erreur : Les deux valeurs doivent être numériques");

        }

      }

      function doTheMath() {
        var first = parseInt(document.mathWiz.numberOne.value);
        var second = parseInt(document.mathWiz.numberTwo.value);
        var operator = document.mathWiz.operator.value;

        switch (operator){
           case "add":
           var answer = first + second;
           break;
          case "subtract":
           var answer = first - second;
           break;
          case "multiply":
           var answer = first * second;
           break;
           case "divide":
```

```
            var answer = first / second;
            break;
          }

          document.mathWiz.theResult.value = answer;
        }
      </script>
    </head>
    <body onload="registerEvents();">
      <div id="formErrors"></div>
      <form name="mathWiz">
        <label>Premier nombre : <input type="number" name="numberOne">
</label><br><br>
        <label>Second nombre : <input type="number" name="numberTwo">
</label><br><br>
        <select name="operator">
          <option value="add"> + </option>
          <option value="subtract"> - </option>
          <option value="multiply"> * </option>
          <option value="divide"> / </option>
        </select>
        <br><br>
        <input type="button" name="operate" value="Faites le calcul !"><br>
<br>
        <label>Résultat : <input type="number" name="theResult"></label>
      </form>
    </body>
</html>
```

La ligne du code du Listing 12.5 qui prend en charge cette validation se trouve dans la fonction `validate ()`. Elle a un aspect un peu étrange :

```
if (/^\d+$/.test (first) & & /^\d+$/.test (second)) {
```

Les caractères entre les barres obliques (/) constituent ce que l'on appelle une *expression régulière*. Une expression régulière est un motif de recherche formé de symboles qui représentent des groupes d'autres symboles. Dans ce cas, nous utilisons une telle expression pour vérifier si les valeurs saisies par l'utilisateur sont toutes deux numériques. Vous en apprendrez plus sur les expressions régulières dans le Chapitre 14.



La validation de la saisie est une application si courante en JavaScript que de nombreuses techniques différentes ont été créées pour réaliser ce type de tâche. Avant de vous mettre à réinventer la roue, je vous suggère de faire une recherche portant sur un critère tel que « validation saisie JavaScript », ou mieux encore (mais en anglais) « open source JavaScript input validation », afin de voir si vous pouvez trouver des librairies de code existantes qui pourraient vous faire gagner beaucoup de temps, tout en vous offrant davantage de fonctionnalités.

Chapitre 13

Travailler avec CSS et les graphismes

DANS CE CHAPITRE :

- **Éditer les styles**
- **Utiliser des images**
- **Exécuter des animations JavaScript**
- **Développer un diaporama**

« Pour avoir du style, commencez par n'en affecter aucun. »

E.B. White, *Les éléments du style*

Une fois que vous avez compris comment manipuler les objets DOM en utilisant JavaScript, les pages Web se transforment de documents statiques en applications dynamiques capables d'interagir avec l'utilisateur, d'évoluer sans avoir à être rechargées, et de délivrer des contenus évolutifs sur de multiples appareils.



N'oubliez pas de visiter le site Web compagnon de ce livre pour y télécharger les exemples de ce chapitre !

Utiliser l'objet Style

L'objet `Style` du modèle DOM est un outil puissant pour rendre une page Web capable de se modifier et de s'adapter en temps réel aux actions de l'utilisateur ou aux conditions courantes du navigateur. L'objet `Style` permet d'accéder aux propriétés de style CSS pour n'importe quel élément ou collection d'éléments dans un document (pour plus d'informations sur les règles de base et la syntaxe de CSS, reportez-vous au Chapitre 1).

Parmi tout ce que vous pouvez faire avec l'objet `Style`, citons notamment :

- Changer les couleurs du texte, par exemple pour mettre en surbrillance des mots-clés dans des champs de recherche.
- Animer un objet lorsque l'utilisateur clique dessus.
- Changer l'encadrement et la couleur d'arrière-plan dans la partie d'un formulaire que l'utilisateur est en train d'éditer.
- Étendre et réduire, ou encore masquer et afficher, différentes parties de la page.
- Créer des bulles ou des cases d'aide qui apparaissent au-dessus du contenu de la page lorsque l'utilisateur clique sur un lien.

L'objet `Style` travaille de la même manière que les autres objets DOM. Il possède un jeu de propriétés qui vous permettent de lire ou de définir les divers aspects d'un élément sélectionné.

Tableau 13.1 : Propriétés courantes de l'objet `Style`, et leurs équivalents CSS.

Propriété	Style CSS	Description
backgroundColor	background-color	Lit ou définit la couleur d'arrière-plan d'un élément
borderWidth	border-width	Définit l'épaisseur des lignes qui forment l'encadrement d'un élément
fontFamily	font-family	Lit ou définit une liste de noms de famille de polices de caractères affectées au texte de l'élément
lineHeight	line-height	Lit ou définit la distance entre les lignes de texte
textAlign	text-align	Lit ou définit l'alignement horizontal du texte dans un élément

Les propriétés de l'objet `Style` reproduisent celles dont vous disposez avec CSS. La différence entre les deux est que le nom de ces propriétés ne comporte pas de tirets pour l'objet `Style`, contrairement à ce qui passe avec CSS.

Le Tableau 13.1 montre quelques-unes des propriétés de l'objet `Style` les plus utilisées, ainsi que les noms des propriétés CSS correspondantes.



Pour consulter une liste complète des propriétés de l'objet `Style`, de même d'ailleurs que de tous les autres objets DOM, voyez le site `http://overapi.com/html-dom`.

Retrouver le style courant d'un élément

L'objet `Style` retourne les styles dits *inline* actuellement affectés à l'élément spécifié. Il ne vous dit donc pas quel est le style effectivement appliqué par le navigateur, car il ne reconnaît pas les styles enregistrés dans des fichiers CSS externes, ou ceux qui sont définis à l'intérieur d'éléments `style`.

En d'autres termes, seuls les styles définis en utilisant JavaScript, ainsi que les styles CSS définis directement, sont récupérables de cette manière.

Pour cette raison, l'objet `Style` n'est pas totalement fiable pour récupérer le style d'un élément. Sur le Listing 13.1, par exemple, l'élément `div` a son propre style défini dans la balise, ainsi que plusieurs règles CSS configurées dans un élément `style`.

LISTING 13.1 : Récupérer le style courant d'un élément (la mauvaise méthode).

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
    <title>Une histoire de style</title>
    <style>
      #myText {
         color:  white;
        background-color: black;
         font-family: Arial;
         margin-bottom: 20px;
      }
      #stylesOutput {
         font-size:  18px;
         font-family: monospace;
      }
    </style>
    <script>
      function getElementStyles(e){
        var colorOutput = "color: " + e.target.style.color;
        var fontSizeOutput = "font-size: " + e.target.style.fontSize;
        document.getElementById("stylesOutput").innerHTML = colorOutput +
"<br>" +
           fontSizeOutput;
      }

    </script>
  </head>
  <body>
    <div id="myText" style="font-size: 26px;"
onclick="getElementStyles(event);">
       Un texte quelconque.</div>
    <div id="stylesOutput"></div>
  </body>
</html>
```

La Figure 13.1 illustre ce qui se passe lorsque vous chargez cette page dans un navigateur et que vous cliquez sur l'élément `div`.



FIGURE 13.1 : Utilisation de l'objet Style pour récupérer le style d'un élément (la mauvaise méthode).

Les deux choses importantes à noter sur les résultats produits par ce script sont les suivantes :

- La valeur de la propriété `color` de l'objet `Style` est vide, bien que la couleur de premier plan de l'objet `div` soit définie comme étant blanche (*white*) dans l'en-tête de la page.
- La valeur de la taille de police est correcte, car la propriété CSS `font-size` est définie dans la balise `div` elle-même.



Les propriétés de l'objet `Style` ne retrouvent que les valeurs des styles *inline* appliquées à un élément.

Une bonne méthode pour obtenir le style courant d'un élément consiste à utiliser `window.getComputedStyle()`, comme l'illustre le Listing 13.2.

LISTING 13.2 : Récupérer le style courant d'un élément (la bonne méthode).

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
    <title>Une autre histoire de style</title>
    <style>
      #myText {
         color:  white;
        background-color: black;
         font-family: Arial;
         margin-bottom: 20px;
      }
     #stylesOutput {
         font-size:  18px;
         font-family: monospace;
      }
    </style>
    <script>
      function getElementStyles(e){

        var computedColor = window.getComputedStyle(e.target).
           getPropertyValue("color");
        var computedSize = window.getComputedStyle(e.target).
           getPropertyValue("font-size");

        var colorOutput = "color: " + computedColor;
        var fontSizeOutput = "font size: " + computedSize;
        document.getElementById("stylesOutput").innerHTML = colorOutput +
"<br>"
          + fontSizeOutput;
      }

    </script>
  </head>
  <body>
    <div id="myText" style="font-size: 26px;"
onclick="getElementStyles(event);">
       Un texte quelconque.</div>
    <div id="stylesOutput"></div>
  </body>
</html>
```

La Figure 13.2 permet de vérifier que le résultat est cette fois correct.



FIGURE 13.2 : Récupérer le style d'un élément (la bonne méthode).

Remarquez que, dans le Listing 13.2, la fonction `getPropertyValue` récupère la propriété CSS (`font-size`) au lieu de la propriété `style` (`fontsize`). La raison tient au fait que le script obtient la valeur de `font-size` directement depuis l'élément, et non *via* l'objet `Style` (qui ne traite que des styles *inline*).

Définir les propriétés de style

Pour définir les propriétés de l'objet `Style`, sélectionnez l'élément pour lequel vous voulez appliquer un nouveau style, puis utilisez la notation point ou avec des crochets droits pour affecter une nouvelle valeur à la propriété voulue.

Pour modifier par exemple le style d'encadrement ayant comme `id` la valeur « borderedSquare », vous pourriez utiliser le code suivant :

```
document.getElementById ("borderedSquare").style.borderWidth = "15px" ;
```

Animer des éléments avec l'objet Style

Vous pouvez utiliser des styles CSS pour contrôler non seulement l'aspect des éléments, mais aussi leur positionnement. En faisant appel à des boucles JavaScript modifiant les propriétés `style`, il est possible de créer assez facilement des animations simples.

Dans le Listing 13.3, une fonction JavaScript déplace un carré sur l'écran en utilisant une boucle `for` pour modifier la propriété CSS `left`.

LISTING 13.3 : Animer un élément avec l'objet Style.

```
<html>
  <head>
    <title>Animation JavaScript</title>
    <style>
      #square {
         width:  100px;
         height:  100px;
        background-color: #333;
         position: absolute;
         left:  0px;
         top:  100px;
      }
    </style>
    <script>
      function moveSquare() {
         for (i=0; i<500; i++){
          document.getElementById("square").style.left = i+"px";
        }
      }
    </script>
  </head>
  <body onload="moveSquare();">
    <div id="square"></div>
  </body>
</html>
```

Si vous exécutez ce code dans un navigateur, vous allez avoir l'impression que l'animation est déjà terminée. En fait, celle-ci est bel et bien exécutée, mais si rapidement que vous ne pouvez pas voir ce qui se passe (à moins d'avoir un ordinateur très lent ou des yeux très rapides).

Pour que l'animation semble se dérouler à une vitesse acceptable par un être humain normalement constitué, il faut insérer une pause entre chaque itération de la boucle. La manière la plus courante de s'y prendre consiste à utiliser la méthode `setTimeout ()` de l'objet `Window`.

La méthode `setTimeout ()` prend deux arguments :

- une fonction, ou un morceau de code à exécuter ;
- un nombre de millièmes de seconde à attendre avant la poursuite de l'exécution.

En plaçant un appel à `setTimeout ()` dans une fonction qui sera appelée de manière récursive, nous pouvons contrôler la vitesse de l'animation (pour en savoir plus sur les fonctions récursives, reportez-vous au Chapitre 7).

Sur le Listing 13.4, le carré est maintenant déplacé par pas de 1pixel à un rythme de 1/100e de seconde. Ce listing propose également plusieurs autres améliorations par rapport au Listing 13.3 :

- Le carré est maintenant cliquable, ce qui déclenche l'animation.

» L'animation du carré est basée sur la position de celui-ci lorsque l'événement 'click' survient. Dans ce cas, le déplacement devient de 100 pixels vers la droite à partir de la position courante.

LISTING 13.4 : Animer un élément avec l'objet Style, la fonction setTimeout () et la récursivité.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
    <title>Animation JavaScript</title>
    <style>
      #square {
         width:  100px;
         height:  100px;
         background-color: #333;
         position: absolute;
         left:  0px;
         top:  100px;
      }
   </style>
    <script>
    // attend que la fenêtre soit totalement chargée
    window.addEventListener('load',initialize,false);

    function initialize(){

      //déplace le carré sur un clic

document.getElementById("square").addEventListener('click',function(e){

      //récupère la position de départ
      var left =
window.getComputedStyle(e.target).getPropertyValue("left");

      //convertit la valeur left en base 10
      left = parseInt(left, 10);

      moveSquare(left,100);

      }, false);

    }

    function moveSquare(left,numMoves) {
      document.getElementById("square").style.left = left+"px";
```

```
      if (numMoves > 0) {
        numMoves--;
         left++;
        setTimeout(moveSquare,10,left,numMoves);
      }else{
         return;
      }
    }
    </script>
  </head>
  <body>
    <div id="square"></div>
  </body>
</html>
```

La Figure 13.3 illustre la sortie produite par le Listing 13.4 lorsqu'il est exécuté dans un navigateur.



Regardez avec attention le code qui enregistre l'événement 'click' pour le carré. Une fonction anonyme est utilisée comme gestionnaire d'événement. Même si cela peut sembler un peu confus en première instance, ce code, si vous le réduisez à ses parties de base, effectue le même travail que la méthode `addEventListener ()`, avec ses trois arguments : le type de l'événement, la fonction à exécuter (en l'occurrence une fonction anonyme), et une valeur booléenne indiquant si le mode capture est utilisé ou non.

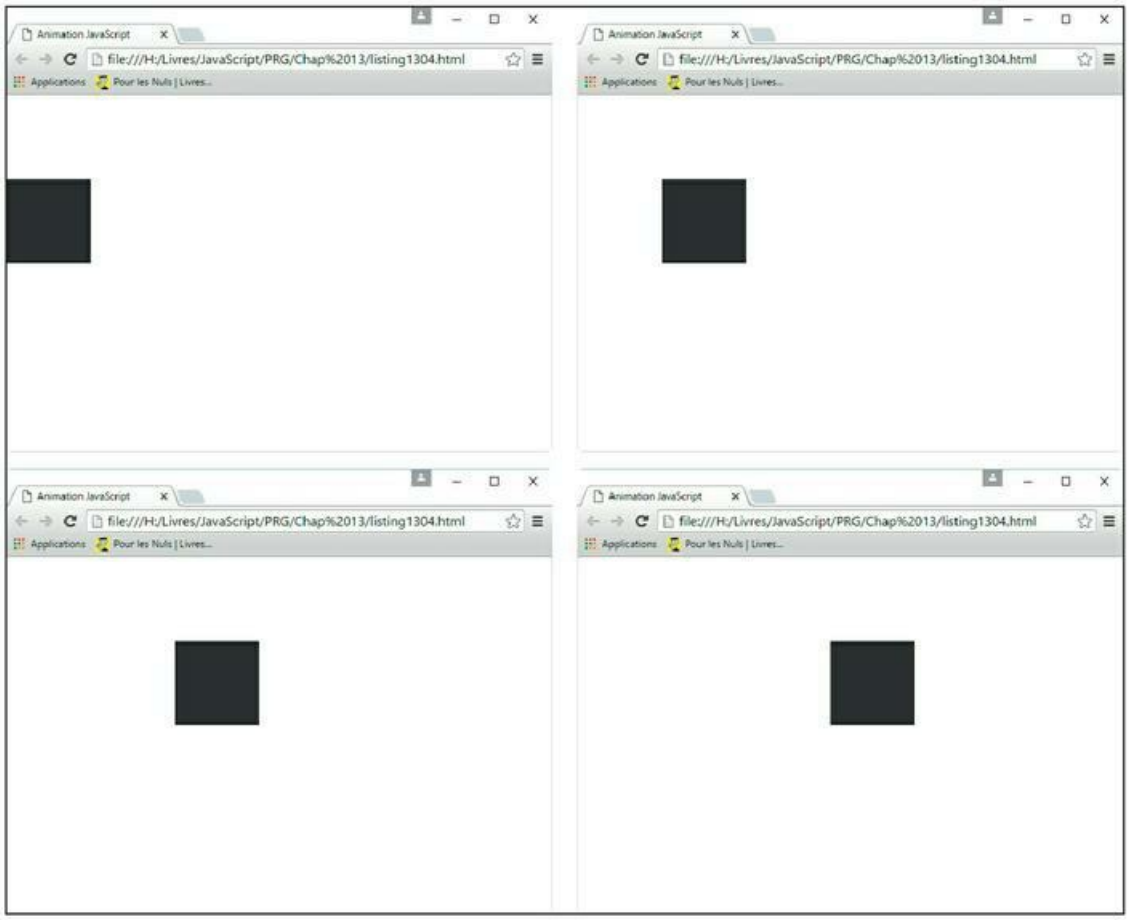


FIGURE 13.3 : JavaScript permet des animations basées sur des événements.

Travailler avec les images

Les éléments HTML `img` sont normalement plutôt statiques, intangibles, à moins bien sûr qu'il ne s'agisse d'animations. Avec JavaScript, il est possible d'ajouter aux images des tas d'effets, comme les redimensionner, les déplacer, les éclairer, leur ajouter des réactions au passage de la souris, et bien d'autres choses encore. Tout cela est rendu possible en manipulant les attributs de l'élément `img` et en changeant des styles CSS.

Utiliser l'objet Image

L'objet `Image` du modèle DOM vous donne accès aux propriétés d'un élément HTML `img`. Vous pouvez ensuite lire ou définir les valeurs de ces propriétés pour gérer tous les attributs valides de l'élément. Les propriétés de l'objet `Image` sont décrites dans le Tableau 13.2.

Tableau 13.2 : Les propriétés de l'objet Image.

Propriété	Description
`alt`	Lit ou définit la valeur de l'attribut `alt` d'une image (un texte de substitution au cas où l'image ne peut pas être affichée)
`complete`	Donne `true` si le navigateur a terminé le chargement de l'image
height	Lit ou définit la valeur de l'attribut `height` (hauteur) d'une image
isMap	Lit ou définit si une image devrait faire partie d'une carte image côté serveur
`naturalHeight`	Lit la hauteur d'origine de l'image
`naturalWidth`	Lit la largeur d'origine de l'image
`src`	Lit ou définit la valeur de l'attribut `src` (source) d'une image
`useMap`	Lit ou définit la valeur de l'attribut `usemap` d'une image
`width`	Lit ou définit la valeur de l'attribut `width` (largeur) d'une image



Les propriétés les plus importantes et les plus utilisées de l'objet `Image` sont `src`, `width` et `height`. Avec ces trois propriétés, vous pouvez facilement échanger des images, provoquer d'étonnants effets de redimensionnement, des boutons qui changent d'aspect au passage de la souris, et bien plus encore !

Créer des boutons avec effet de survol (rollover)

Un bouton *rollover* (c'est l'appellation généralement employée) est un bouton qui change d'aspect lorsque le pointeur de la souris le survole. C'est par exemple un bon moyen pour indiquer à l'utilisateur qu'une image peut être cliquée. Vous pouvez aussi utiliser de tels boutons pour révéler plus d'informations sur ce qui se passe dans le cas où l'utilisateur clique dessus. Et les motivations purement esthétiques sont tout autant estimables (à condition de ne pas tomber dans le péché de surcharge). Certains programmeurs aiment à placer ainsi dans leurs sites Web ce que l'on appelle des *œufs de Pâques* (*easter eggs* en anglais), et qui sont souvent des sortes de boutons rollover cachés dans la page qui déclenchent par exemple une animation, voire même affichent une page de bonus, lorsque l'utilisateur clique dessus.

Vous pouvez parfaitement créer un effet de rollover avec CSS, mais pour obtenir des résultats plus sophistiqués, il est nécessaire d'en passer par JavaScript, ou par une combinaison de JavaScript et de CSS.



L'exemple du Listing 13.5 montre comment créer un effet de rollover simple avec une image en JavaScript.

LISTING 13.5 : Un effet de rollover sur une image.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
    <title>Image avec effet Rollover</title>
    <script>

    function swapImage(imgToSwap){
      imgToSwap.src = "button2.png";
      imgToSwap.alt = "Vous survolez mon bouton avec votre souris !";
    }
    function swapBack(imgToSwap){
      imgToSwap.src = "button1.png";
      imgToSwap.alt = "Survolez moi !";
    }

    </script>
  </head>
  <body>
    <img src="button1.png" id="myButton" onmouseover="swapImage(this);"
onmouseout=
        "swapBack(this);" alt="Survolez moi !">
  </body>
</html>
```



Pour que le Listing 13.5 fonctionne correctement, vous devez disposer d'images appelées `button1.png` et `button2.png` placées dans le même dossier que le fichier HTML. Vous pouvez utiliser celles du site compagnon de ce livre, ou bien créer les vôtres.

Des images que la souris fait grandir

Un autre tuc qui permet de rendre les sites Web plus attractifs pour l'utilisateur consiste à augmenter légèrement la taille d'un bouton image lorsqu'il est survolé par le pointeur de la souris, ce qui donne une sensation de relief. C'est aussi une manière d'indiquer que l'image est cliquable, tout en fournissant une interactivité supplémentaire.

Le Listing 13.6 modifie le code du Listing 13.5 en agrandissant l'image du bouton de 5 % en réponse aux événements `mouseover`.



Faites attention à ne pas trop agrandir l'image. Dans ce cas, sa qualité pourrait en être fortement altérée, ce qui rendrait l'effet désagréable pour l'utilisateur.

LISTING 13.6 : Augmenter la taille de l'image au survol de la souris.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
  <title>Redimensionnement de l'image</title>
  <script>

    function growImage(imgToGrow){
      imgToGrow.width += imgToGrow.width * .05;
      imgToGrow.height += imgToGrow.width * .05;
    }
    function restoreImage(imgToShrink){
      imgToShrink.width = imgToShrink.naturalWidth;
      imgToShrink.height = imgToShrink.naturalHeight;
    }

  </script>
</head>
<body>
  <img src="button1.png" id="myButton" onmouseover="growImage(this);"
onmouseout=
    "restoreImage(this);" alt="Survolez moi !">
</body>
</html>
```



Les Listings 13.5 et 13.6 utilisent la méthode de définition des événements *inline* (dans la balise elle-même). Cette technique n'est pas idéale pour le développement d'applications Web, mais elle est fréquemment employée pour des effets de survol simples qui sont des éléments d'interface, et non de véritables fonctionnalités de l'application.

Créer un diaporama

Les *diaporamas* (ou *carrousels*) sont un procédé courant lorsque l'on veut afficher une série d'images dans un même espace du site. Même si on les rencontre le plus souvent sur les pages d'accueil, ils peuvent donner du dynamisme n'importe où dans vos sites.

Les diaporamas sont souvent associés à des effets de transition lors du passage d'une image à une autre. Ces transitions sont généralement créées en faisant appel à une bibliothèque de fonctions JavaScript, comme *jQuery*. Mais il est aussi possible de se contenter de JavaScript, CSS et DOM. Pour rester simple, le Listing 13.7 se contente de passer d'une image à l'autre sans transition.

LISTING 13.7 : Un diaporama construit avec JavaScript et CSS.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
  <title>Diaporama JavaScript</title>
```

```
  <style>
    #carousel {
      position: absolute;
       width:  800px;
       height:  400px;
       top:  100px;
       left:  100px;
       display:  hidden;
    }

  </style>
  <script>
    var slides = [
      "<div id='slide1'>ma première photo<br><img src='image1.jpg'>
</div>",
      "<div id='slide2'>ma deuxième photo<br><img src='image2.jpg'>
</div>",
      "<div id='slide3'>ma troisième photo<br><img src='image3.jpg'>
</div>"];

    var currentSlide = 0;
    var numberOfSlides = slides.length-1;

    window.addEventListener("load",loader,false);

    function loader(){
      changeImage();
    }

    function changeImage(){
      console.log("Fonction changeImage");
      if (currentSlide > numberOfSlides){
        currentSlide = 0;
      }

      document.getElementById("carousel").innerHTML=slides[currentSlide];

      console.log(‘affichage de la photo' + currentSlide + " sur " +
             numberOfSlides);
      currentSlide++;

      setTimeout(changeImage,1000);
    }

  </script>
```

```
</head>
<body>
  <div id="carousel"></div>
</body>
</html>
```

Utiliser les propriétés d'animation de l'objet Style

CSS3 et l'objet `Style` du modèle DOM possèdent des propriétés qui simplifient l'animation des éléments. Utilisées ensemble, ces propriétés peuvent vous permettre de créer des animations amusantes ou intéressantes avec un minimum d'efforts. Les propriétés d'animation de l'objet Style sont décrites dans le Tableau 13.3.

Tableau 13.3 : Propriétés liées à l'animation de l'objet Style.

Propriété	Description
`animation`	Définit simultanément toutes les propriétés de l'animation à l'exception de `animationPlayState`
`animationDelay`	Lit ou définit un délai avant le lancement de l'application
`animationDirection`	Lit ou définit si l'animation devrait être jouée en sens inverse pour certains des cycles, ou tous
`animationDuration`	Lit ou définit la durée d'un cycle d'animation
`animationFillMode`	Lit ou définit les valeurs qui sont appliquées en dehors des cycles d'animation
`animationIterationCount`	Lit ou définit le nombre d'exécutions de l'animation
`animationName`	Lit ou définit une liste d'animations
`animationTimingFunction`	Lit ou définit la courbe de vitesse décrivant la manière dont l'animation devrait progresser dans le temps
`animationPlayState`	Lit ou définit si l'animation est en cours d'exécution ou en pause

Sur le Listing 13.8, une animation simple a été créée en utilisant CSS. La durée de celle-ci, ainsi que ses principales propriétés, sont définies en CSS dans la section `style`, puis JavaScript prend le relais pour mettre en pause ou reprendre l'animation. Avec un peu de créativité, vous trouverez de multiples manières de contrôler cette animation avec JavaScript.



Les animations CSS3 sont relativement nouvelles, ce qui implique qu'elles ne sont pas prises en charge de la même manière par tous les navigateurs. Du fait qu'il s'agit d'une technologie toujours considérée comme étant expérimentale, certains navigateurs nécessitent un préfixe spécial avant le nom des propriétés de l'animation.

LISTING 13.8 : Contrôler une animation CSS3 avec JavaScript.

```
<!DOCTYPE html>
```

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
<style>
#words {

    position: relative;
    width:  300px;
    height:  200px;
    text-align: center;
    padding-top: 20px;
    font-family: Arial;
    border-radius: 6px;
    color:  white;

    /* Chrome, Safari, Opera */
    -webkit-animation-name: movewords;
    -webkit-animation-duration: 6s;
    -webkit-animation-timing-function: linear;
    -webkit-animation-delay: 0s;
    -webkit-animation-iteration-count: infinite;
    -webkit-animation-direction: alternate;
    -webkit-animation-play-state: running;
    /* Standard syntax */
    animation-name: movewords;
    animation-duration: 6s;
    animation-timing-function: linear;
    animation-delay: 0s;
    animation-iteration-count: infinite;
    animation-direction: alternate;
    animation-play-state: running;
}

/* Chrome, Safari, Opera */
@-webkit-keyframes movewords {
    0%   {background:red;  left:100px;  top:0px;}
    25%  {background:blue;  left:200px;  top:100px;}
    50%  {background:blue;  left:300px;  top:200px;}
    75%  {background:blue;  left:200px;  top:200px;}
    100% {background:red;  left:100px;  top:0px;}
}

/* Standard syntax */
@keyframes movewords {
    0%   {background:red;  left:100px;  top:0px;}
    25%  {background:blue;  left:200px;  top:100px;}
```

```
    50%  {background:blue;  left:300px;  top:200px;}
    75%  {background:blue;  left:200px;  top:200px;}
    100% {background:red;  left:100px;  top:0px;}
}
</style>
<script>

    window.addEventListener("load",registerEvents,false);

    function registerEvents(e){

document.getElementById("stop").addEventListener("click",stopAni,false);

document.getElementById("go").addEventListener("click",startAni,false);
    }
    function stopAni(){
        document.getElementById("words").style.webkitAnimationPlayState =
"paused";
        document.getElementById("words").style.AnimationPlayState =
"paused";

    }
    function startAni(){
        document.getElementById("words").style.webkitAnimationPlayState =
"running";
        document.getElementById("words").style.AnimationPlayState =
"running";
    }

</script>

</head>
<body>

<h1 id="words">On se bouge</h1>

<button type="button" id="stop">Pause !</button>
<button type="button" id="go">Action !</button>

</body>
</html>
```

Au-delà des bases

DANS CETTE PARTIE...

Apprendre à effectuer des recherches avec les expressions régulières

Découvrir comme utiliser les fonctions de rappels et les fermetures

Toujours plus loin avec Ajax et JSON

Chapitre 14

Effectuer des recherches avec des expressions régulières

DANS CE CHAPITRE :

- **Trouver des motifs avec des expressions régulières**
- **Écrire des expressions régulières**
- **Utiliser des expressions régulières en JavaScript**

« Créer des problèmes est facile. Nous le faisons tout le temps. Trouver des solutions définitives et qui produisent de bons résultats, cela demande beaucoup d'efforts et de soin. »

Henry Rollins

Les expressions régulières sont un outil puissant dans de nombreux langages de programmation. Elles permettent de trouver et de modifier du texte dans les documents en fonction d'un certain motif. La syntaxe des expressions régulières peut être intimidante au premier abord, mais, lorsque vous arrivez à les maîtriser, elles vous permettent de faire ce que vous voulez avec les textes.



N'oubliez pas de visiter le site Web compagnon de ce livre pour y télécharger les exemples de ce chapitre !

Faire des recherches avec les expressions régulières

Les expressions régulières sont une technique permettant de retrouver des motifs ou des combinaisons de caractères dans les chaînes.

Parmi les exemples d'utilisation de ces expressions régulières, mentionnons-en quelques-uns :

- Vérifier qu'un utilisateur a bien saisi une adresse de messagerie en respectant le format voulu.
- Trouver et remplacer toutes les occurrences du nom d'une personne dans un article.
- Localiser dans un livre les mots en majuscules au milieu des phrases.
- Trouver des chaînes numériques semblant être des numéros de téléphone.

Voici à quoi une expression régulière peut ressembler :

```
^ ((\ (\d{3}\) ?) | (\d{3}-)) ? \d{3}-\d{4}$
```

Plutôt impressionnant, non ? Mais ne paniquez pas. Vous disposerez bientôt des outils nécessaires pour décoder cette expression. Vous comprendrez alors qu'elle est formée de manière à retrouver des numéros de téléphone au format généralement employé aux USA, comme dans :

```
(555) 555-5555
```

Bien sûr, les expressions régulières peuvent être plus simples que ci-dessus. Le Listing 14.1 montre un exemple d'utilisation d'une expression régulière, le résultat de son exécution étant illustré sur la Figure 14.1.

LISTING 14.1 : Est-ce que la chaîne contient le mot « JavaScript » ?

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
    <title>Recherche en JavaScript</title>
    <script>
    window.addEventListener("load",registerEvents,false);

    function registerEvents(e){

document.getElementById("ask").addEventListener("click",findAnswer,false);

    }

    function findAnswer(){
      var question = document.getElementById("userQuestion").value;
     var re = new RegExp("JavaScript");
      if (re.test(question)===true){
        document.getElementById("answer").innerHTML = "Une question sur
         JavaScript ? Voyez le livre Coder avec JavaScript pour les Nuls
par Chris
          Minnick et Eva Holland";
        console.log("JavaScript!");
      }
    }
    </script>
  </head>
  <body>
    <form id="userInput">
      <label>Entrez votre question :
      <textarea id="userQuestion"></textarea>
      </label>
      <button id="ask" type="button">Obtenir une réponse</button>
    </form>
    <div id="answer"></div>
  </body>
</html>
```


Recherche en JavaScript
file:///H:/Livres/JavaScript/PRG/Chap%2014/listing1401.html
Applications
Pour les Nuls | Livres...
Comment devenir un bon programmeur en JavaScript ?
Entrez votre question :
Obtenir une réponse
Une question sur JavaScript ? Voyez le livre Coder avec JavaScript pour les Nuls par Chris Minnick et Eva Holland

FIGURE 14.1 : Le code du Listing 14.1 en action.

Écrire des expressions régulières

Avant de pouvoir utiliser une expression régulière, vous devez créer un objet qui contient cette expression. Vous disposez pour cela de deux méthodes :

- Utiliser un littéral.
- *Via* la fonction de construction de l'objet `RegExp`.

Utiliser l'objet RegExp

Lorsque vous définissez une expression régulière en appelant la fonction de constructeur de l'objet `RegExp`, l'objet résultant est créé lors de l'exécution et lorsque le script est chargé. Vous devriez utiliser cette fonction si vous ne connaissez pas la valeur de l'expression régulière lorsque vous écrivez le script. Par exemple, vous pourriez demander à l'utilisateur de saisir une expression, ou bien encore vous pourriez obtenir celle-ci depuis un fichier extérieur, ou encore calculer cette expression lorsque le script est exécuté.

Le programme du Listing 14.2 crée une expression régulière à partir d'une lettre prise aléatoirement, et elle demande ensuite à l'utilisateur de saisir une phrase. Une fois le formulaire soumis, le programme calcule le nombre d'instances de la lettre aléatoire dans la réponse soumise par l'utilisateur.

LISTING 14.2 : Créer des expressions régulières au moment de l'exécution avec l'objet RegExp.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
    <title>Compter le nombre de lettres</title>
    <script>
      window.addEventListener('load',loader,false);

      //produit une lettre aléatoire
      var letter = String.fromCharCode(97 + Math.floor(Math.random() *
26));
```

```
      //crée une  expression régulière en utilisant la lettre. Configure
l'option g
        pour trouver toutes les occurrences.
      var re = new RegExp(letter,'g');

      function loader(e){

document.getElementById("getText").addEventListener('submit',countLetter,
         false);
      }

      function countLetter(e){
        e.preventDefault();
        document.getElementById("results").innerHTML = "La lettre secrète
était "
          + letter +".";
        var userText = document.getElementById("userWords").value;
        var matches = userText.match(re);

        if (matches){
        var count = matches.length;
        } else {
        var count = 0;
        }
       document.getElementById("results").innerHTML += " Vous avez saisi
la lettre
          secrète " + count + " fois.";
      }

    </script>
  </head>
  <body>
    <form id="getText">
      <p>Je pense à une lettre ! Tapez une phrase, et je vous dirais
combien de fois
        votre phrase utilise ma lettre secrète !</p>
      <input type="text" name="userWords" id="userWords">
      <input type="submit" name="submit">
    </form>
    <div id="results"></div>
  </body>
</html>
```

La Figure 14.2 illustre l'exécution du code du Listing 14.2.

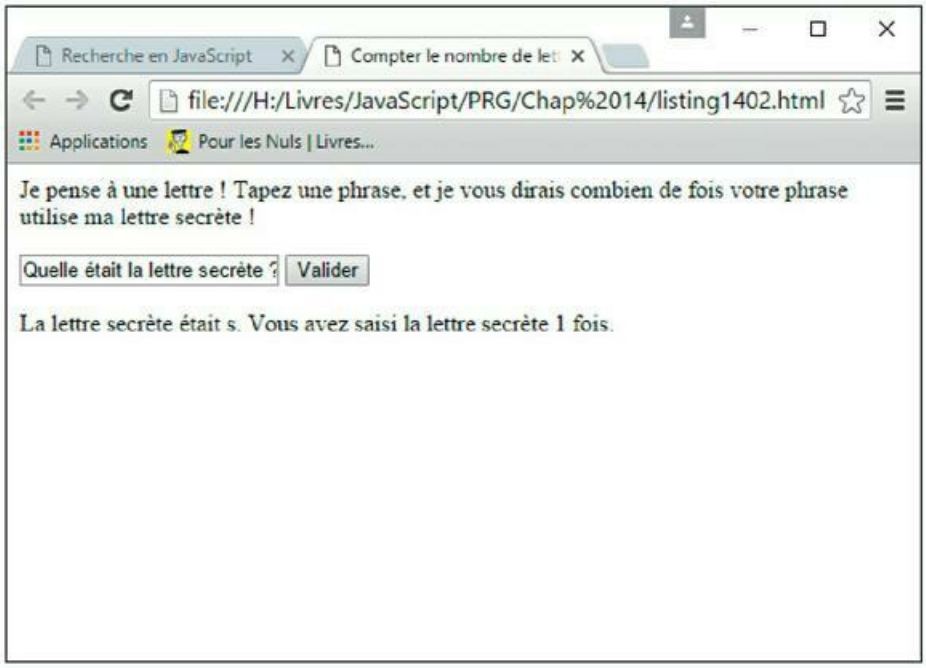

Recherche en JavaScript
Compter le nombre de let
file:///H:/Livres/JavaScript/PRG/Chap%2014/listing1402.html
Applications
Pour les Nuls | Livres...
Je pense à une lettre ! Tapez une phrase, et je vous dirais combien de fois votre phrase utilise ma lettre secrète !
Quelle était la lettre secrète ?
Valider
La lettre secrète était s. Vous avez saisi la lettre secrète 1 fois.

FIGURE 14.2 : Un petit jeu de lettre.

Définir des expressions régulières littérales

Pour créer une expression régulière littérale, vous entourez la valeur de cette expression par des barres obliques au lieu d'apostrophes.

Par exemple :

```
var monExpressionReg = /JavaScript/ ;
```

Ces expressions littérales sont compilées par le navigateur lorsque le script est téléchargé, et restent identiques au cours de la durée de vie du script. Le résultat est que ce type d'expression régulière offre de meilleures performances tant que les expressions ne sont pas modifiées.

L'exemple précédent utilise une expression régulière pour rechercher une correspondance exacte avec la chaîne "JavaScript". Cependant, dans des applications ou programmes réels, vous auriez sans doute besoin d'autoriser certaines variations quant à l'orthographe exacte du mot, par exemple :

- javascript
- Javascript
- java script
- JS
- js

Il pourrait même y avoir quelques variantes encore plus exotiques. L'une des choses les plus étonnantes et les plus frustrantes dans ce cas, c'est que l'inattendu est ce à quoi vous pouvez vous attendre le plus ! Pour détecter toutes ces variations sur le même thème, vous pouvez utiliser des expressions régulières encore plus sophistiquées pour rechercher des motifs ou jeux de caractères, plutôt que des chaînes littérales.

Le code qui suit étend le critère de recherche de manière à localiser aussi bien "JavaScript" que "Javascript" ou encore "javascript" :

```
var monExpressionReg = /[Jj]ava[Ss]cript/ ;
```

Tout cela peut sembler un peu lointain. Mais, si vous y regardez d'un peu plus près, vous constaterez que les choses sont finalement assez simples. La partie qui se trouve entre les crochets droits est une expression régulière qui définit un ensemble de caractères, et tout ce qui correspond à ce qui se trouve entre ces crochets sera vérifié. En écrivant `[Jj]`, nous nous assurons que l'emploi de majuscules comme de minuscule sera considéré comme valide, du moins pour ce qui concerne la lettre `J`.

Tester des expressions régulières

Lorsque vous écrivez des expressions régulières, il peut être utile de disposer d'un moyen commode de les tester afin de vous assurer qu'elles font bien ce que vous avez prévu. Nombre de sites Web et d'outils vous permettent d'effectuer ces tests. Parmi ces sites, mentionnons en particulier `http://regex101.com`. Pour l'utiliser, saisissez votre expression régulière dans le champ du haut, puis tapez un texte à votre convenance dans le second champ. Le site va mettre en surbrillance les correspondances qu'il trouve entre votre expression et le texte à tester.

La Figure 14.3 illustre le résultat produit par `regex101.com` sur l'expression régulière de l'exemple précédent.

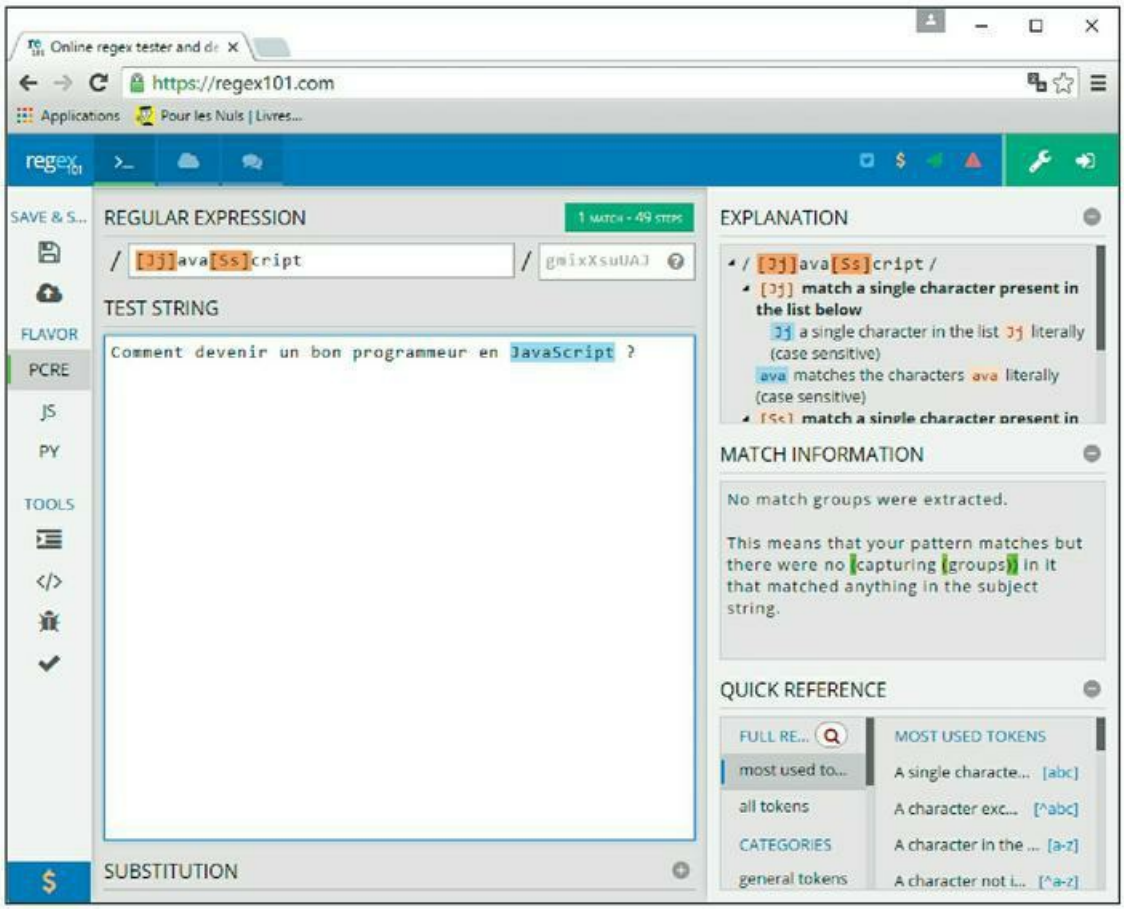

https://regex101.com
REGULAR EXPRESSION
/ [Jj]ava[Ss]cript /
TEST STRING
Comment devenir un bon programmeur en JavaScript ?
EXPLANATION
MATCH INFORMATION
No match groups were extracted.
This means that your pattern matches but there were no capturing groups in it that matched anything in the subject string.
QUICK REFERENCE
SUBSTITUTION

FIGURE 14.3 : Le site regex101.com permet de tester une expression régulière.

Caractères spéciaux dans les expressions régulières

Les expressions régulières vous permettent de rechercher des nombres dans des chaînes, de même que des lettres, des groupes de lettres, des caractères répétitifs, et bien plus encore.

Pour créer des motifs de recherche complexes, vous pouvez utiliser les caractères spéciaux reconnus par les expressions régulières. Les plus courants sont listés dans le Tableau 14.1.

Tableau 14.1 : Caractères spéciaux dans les expressions régulières.

Caractère spécial	Signification
\	Indique si le caractère qui suit devrait être traité comme caractère spécial ou comme littéral. Si le caractère suivant est également un caractère spécial, \ demande à ce qu'il soit traité littéralement.
^	Trouve le début de la saisie
$	Trouve la fin de la saisie
*	Trouve le caractère précédent zéro fois ou plus
+	Trouve le caractère précédent une fois ou plus
?	Trouve le caractère précédent zéro ou une fois
.	Trouve n'importe quel caractère unique à l'exception du changement de ligne
x|y	Trouve soit x soit y
{n}	Trouve exactement *n* occurrences du caractère précédent
[xyz]	Trouve n'importe lequel des caractères entre crochets
[^xyz]	Trouve n'importe quel caractère autre que ceux placés entre crochets
[\b]	Trouve une espace
\b	Trouve une limite de mot (attention : les caractères accentués sont traités comme des limites de mots)
\B	Ne trouve pas de limite de mot
\d	Trouve un caractère numérique
\D	Trouve n'importe quel caractère non numérique
\n	Trouve un retour à la ligne
\s	Trouve un caractère blanc (espace, tabulation, retour à la ligne, *etc.*)
\S	Trouve un caractère qui n'est pas un caractère blanc
\t	Trouve une tabulation
\w	Trouve un caractère alphanumérique, y compris le trait de soulignement
\W	Trouve un caractère qui n'est pas un caractère « de mot »

Utiliser des modificateurs

Les modificateurs servent à changer globalement plusieurs paramètres de recherche. Pour cela, vous passez ces modificateurs comme second argument du constructeur `RegExp()` lorsque vous créez votre expression régulière, ou vous les placez après la barre oblique / finale dans une expression définie sous forme de littéral.

Les trois modificateurs sont :

- `g` (global) : Indique que la chaîne toute entière devrait être recherchée, et pas seulement jusqu'à ce que la première occurrence soit trouvée.
- `i` (insensible à la casse) : Indique que les majuscules et les minuscules doivent être indifférenciées.
- `m` (multiligne) : Effectue une recherche multiligne. Par exemple, lorsque les caractères spéciaux ^ (début) et $ (fin) sont utilisés, le modificateur traite chaque nouvelle ligne comme un nouveau début/une nouvelle fin, au lieu de prendre en compte seulement le début et la fin de l'entrée.

Ainsi, l'expression régulière suivante recherchera toutes les variantes du mot JavaScript dans la totalité du document :

```
/javascript/ig
```

Coder avec des expressions régulières

Les expressions régulières disposent de leurs propres méthodes, ainsi que d'un sous-ensemble des fonctions de chaîne (voyez à ce sujet le Chapitre 3).

Les méthodes des expressions régulières sont les suivantes :

- `test` : Teste une correspondance. Renvoie `true` en cas de succès, `false` sinon.
- `exec` : Teste une correspondance. Retourne un tableau d'informations sur cette correspondance.

Si vous avez uniquement besoin de savoir si l'expression régulière trouve une correspondance pour le critère de recherche qu'elle définit, vous devriez utiliser la méthode `test`. Par contre, si vous voulez savoir où se trouve cette correspondance (ou ces correspondances) dans une chaîne, quel est le nombre de correspondances, et le texte correspondant, vous devriez faire appel à la méthode `exec`.

Les fonctions de chaîne qui peuvent être utilisées avec des expressions régulières sont décrites dans le Tableau 14.2.

Tableau 14.2 : Fonctions de chaîne pour les expressions régulières.

Fonction	Utilisation
`match`	Recherche une correspondance ou l'expression régulière dans une chaîne. Retourne un tableau d'informations sur la correspondance, ou `null` en cas d'échec.
`search`	Teste une correspondance dans une chaîne. En cas de succès, renvoie l'indice de la correspondance, ou -1 en cas d'échec.
`replace`	Recherche une correspondance dans une chaîne, et remplace celle-ci par la chaîne spécifiée.
`split`	Partage une chaîne en un tableau de sous-chaînes en utilisant une expression régulière ou une chaîne fixée.

La vérification d'une adresse de messagerie est un bon exemple, et de surcroît étonnamment complexe, d'utilisation des expressions régulières. Toute adresse de messagerie valide doit se conformer à certaines règles, en particulier celles-ci :

- L'adresse doit contenir le symbole @.
- Le symbole @ doit être précédé et suivi de caractères.
- Il doit y avoir au moins un séparateur de groupe après le symbole @.

Il y a encore d'autres règles, mais les choses se compliquent assez rapidement si vous commencez à fouiller dans les détails, par exemple le fait que des espaces peuvent être autorisées dans certains cas, de même que l'emploi de caractères internationaux.

Pour un formulaire qui n'a pas besoin d'entrer dans ces considérations sophistiquées, il suffit généralement de demander à l'utilisateur de saisir son adresse de messagerie dans un champ de texte, puis d'effectuer des vérifications sur la base des règles énoncées ci-dessus. Cela devrait permettre d'éliminer la plupart des erreurs.

Le Listing 14.3 illustre cela. Une fois que l'utilisateur a saisi une adresse de messagerie et a cliqué sur le bouton de validation, le script teste cette adresse à l'aide de l'expression régulière littérale suivante :

```
/\b[A-Z0-9._ % +-]+@[A-Z0-9.-]+\.[A-Z]{2,4}\b/i
```

Cette expression débute par `\b`, le caractère spécial de recherche de limite de nouveau mot. Il est suivi du motif suivant :

```
[A-Z0-9._ % +-]+
```

Ce motif recherche une ou plusieurs combinaisons de lettres ou de chiffres, ainsi que de traits de soulignement, de signes de pourcentage ou de tirets. Vient ensuite :

```
@[A-Z0-9.-]+
```

Cette partie rend obligatoire le séparateur d'adresse @, à nouveau suivi d'une combinaison de lettres, de chiffres et de tirets. L'expression se termine par :

```
\.[A-Z]{2,4}\b/i
```

La fin de l'expression régulière recherche une chaîne commençant par un point (le séparateur dans le nom de domaine de l'adresse de messagerie), et suivie de deux à quatre caractères de longueur (par exemple `fr`, `com`, `net` ou encore `org`). La recherche se termine avec le caractère spécial `\b` (fin de mot). Le modificateur `/i` indique que les caractères peuvent être aussi bien saisis en minuscules qu'en majuscules.

Si une correspondance est trouvée, cela signifie que les données saisies par l'utilisateur ont passé le test, ce qui est signalé par le message `"Valide ! "`.



Avec l'extension actuelle des noms de domaine Internet, l'expression rationnelle utilisée ici pourrait ne pas être suffisante. Par exemple, le suffixe peut maintenant dépasser les quatre caractères, voire comporter éventuellement deux points ou même plus (comme dans `.gouv.fr` ou encore `.finances.gouv.fr`). À charge pour vous donc d'améliorer notre expression régulière pour prendre ces situations en compte…

LISTING 14.3 : Un script de validation d'adresse de messagerie.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
  <title>Validateur de messagerie</title>
  <script>
    window.addEventListener('load',loader,false);
    function loader(e){
      e.preventDefault();

document.getElementById('emailinput').addEventListener('submit',validateEma

il,false);
    }

    function validateEmail(e) {
        var re = /\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}\b/i;
        if (re.test(e.target.yourEmail.value)) {
           alert("Valide  !");
        } else {
           alert("Invalide !");
        }
    }

  </script>
</head>
<body>
  <form id="emailinput">
    <label>Entrez une adresse de messagerie :
      <input type="text" id="yourEmail">
    </label>
    <input type="submit" value="Valider" id="validate">
  </form>
</body>
</html>
```

La Figure 14.4 illustre l'exécution de ce code.

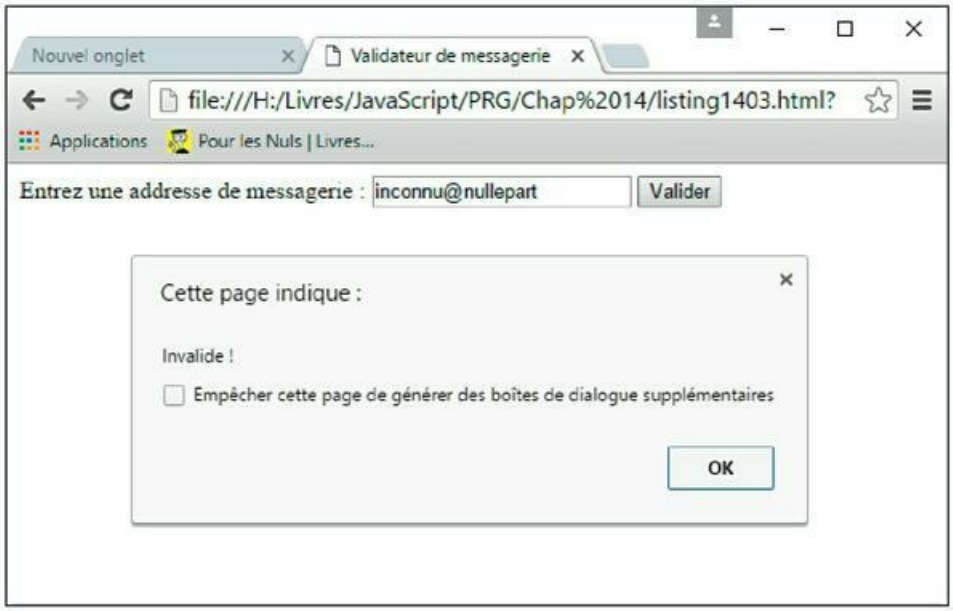


FIGURE 14.4 : Utiliser une expression régulière dans un script de validation d'adresse de messagerie.

Chapitre 15

Comprendre les fonctions de rappel et les fermetures

DANS CE CHAPITRE :

- **Comprendre les fonctions de rappel**
- **Utiliser des fonctions de rappel**
- **Créer des fermetures**

« Hier est parti pour toujours. Demain ne viendra peut-être jamais. Seul aujourd'hui t'appartient. »

Anonyme

Les fonctions de rappel (*callbacks*) et les fermetures (*closures*) sont deux des techniques les plus utiles et les plus largement utilisées en JavaScript. Dans ce chapitre, vous allez apprendre pourquoi et comment passer des fonctions en tant que paramètres à d'autres fonctions.



N'oubliez pas de visiter le site Web compagnon de ce livre pour y télécharger les exemples de ce chapitre !

C'est quoi, les fonctions de rappel ?



Les fonctions de JavaScript sont des objets. Cette phrase est la clé pour comprendre la plupart des techniques les plus avancées de JavaScript, y compris donc les fonctions de rappel.

Les fonctions, comme tous les autres objets, peuvent être affectées à des variables, être passées en argument à d'autres fonctions, ou encore être créées et retournées à partir de fonctions.

Passer des fonctions comme argument

Une *fonction de rappel* (ou *callback*) est une fonction qui est passée en tant qu'argument à une autre fonction. Il s'agit d'une technique qui est possible en JavaScript du fait que les fonctions sont considérées comme des objets.

Les objets « fonction » contiennent une chaîne qui est le code de la fonction. Lorsque vous appelez une fonction en donnant son nom suivi de parenthèses, vous lui demandez d'exécuter son code. Par contre, lorsque vous nommez une fonction ou que vous la passez en argument sans spécifier les parenthèses, son code n'est pas exécuté.

Le Chapitre 11 contient des exemples de fonctions de rappel utilisées avec la méthode `addEventListener`, comme dans :

```
document.addEventListener ('click', faireQuelqueChose, false) ;
```

Cette méthode prend comme arguments un événement (`click`) et un objet `Function` (`faireQuelqueChose`). La fonction de rappel n'est pas exécutée tout de suite, mais uniquement quand l'événement est déclenché.

Écrire des fonctions avec des rappels

Voici un exemple simple de fonction, `doMath`, qui accepte en argument une fonction de rappel :

```
function doMath(number1,number2,callback){
      var result = callback(number1,number2);
      document.write ("Le résultat est : " + result);
}
```

Il s'agit ici d'une fonction générique qui peut retourner le résultat produit par n'importe quel opérateur mathématique demandant deux opérandes. La fonction de rappel que vous lui passez spécifie les opérations qui doivent être effectuées.

Pour appeler notre fonction `doMath`, il faut lui passer deux arguments numériques, puis une fonction en tant que troisième argument. Par exemple :

```
doMath(5,2,function(number1,number2){
  var calculation = number1 * number2 / 6;
  return calculation;
});
```

Le Listing 15.1 propose un exemple d'application dans lequel la fonction `doMath` est appelée plusieurs fois avec différentes fonctions de rappel.

LISTING 15.1 : Appeler une fonction avec différentes fonctions de rappel.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
  <title>Introduction à la fonction doMath</title>
  <script>
    function doMath(number1,number2,callback){

      var result = callback(number1,number2);
      document.getElementById("theResult").innerHTML += ("Le résultat est
: " +
result + "<br>");

    }

    document.addEventListener('DOMContentLoaded', function() {

      doMath(5,2,function(number1,number2){
        var calculation = number1 * number2;
        return calculation;
        });

      doMath(10,3,function(number1,number2){
        var calculation = number1 / number2;
        return calculation;
        });

      doMath(81,9,function(number1,number2){
        var calculation = number1 % number2;
        return calculation;
        });

    }, false);
  </script>
</head>
<body>
  <h1>Faites le calcul</h1>
  <div id="theResult"></div>
</body>
</html>
```

Le résultat produit par l'exécution de ce code est illustré sur la Figure 15.1.

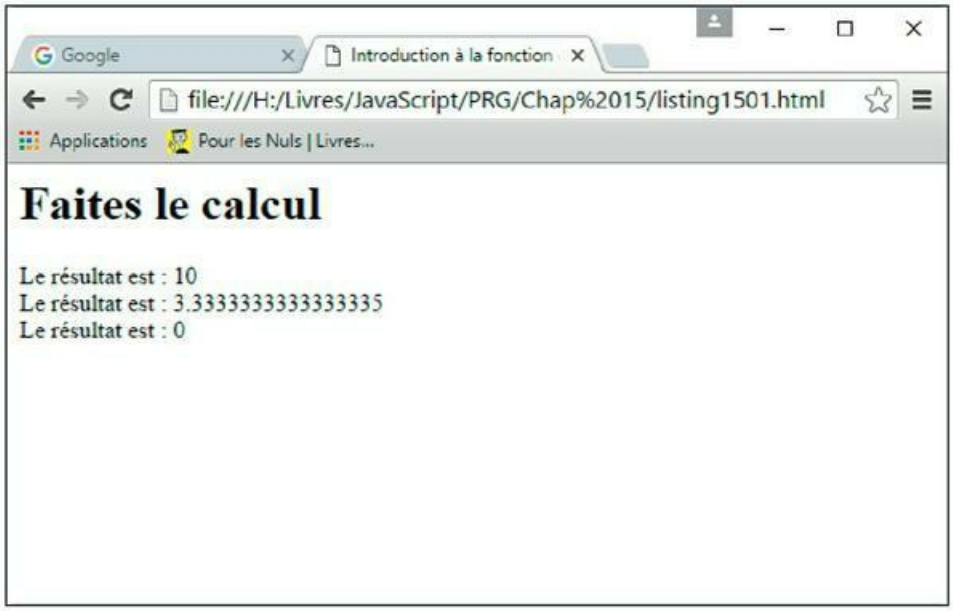


FIGURE 15.1 : Effectuer des calculs en utilisant des fonctions de rappel.

Utiliser des fonctions de rappel nommées

Dans la section précédente, toutes les fonctions de rappel sont anonymes. Mais il est aussi possible de définir des fonctions nommées, et de passer leur nom en tant que fonction de rappel.



Une fonction anonyme est une fonction qui est définie sans lui donner de nom. Revoyez à ce sujet le Chapitre 7.

Utiliser des fonctions nommées comme rappels peut aider à la compréhension du code, ce qui n'est pas forcément le cas avec les fonctions anonymes. Le Listing 15.2 montre un exemple d'utilisation de fonction nommée en tant que rappel. Il propose également deux améliorations par rapport au Listing 15.1 :

- Un test a été ajouté à la fonction `doMath` afin de s'assurer que l'argument `callback` est effectivement une fonction.
- Le code de la fonction `callback` est affiché avant de le montrer.

LISTING 15.2 : Utiliser des fonctions nommées en tant que rappels.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
  <title>doMath avec des fonctions nommées</title>
  <script>
  function doMath(number1,number2,callback){

    if (typeof callback === "function") {

    var result = callback(number1,number2);
    document.getElementById("theResult").innerHTML +=
(callback.toString()
```

```
      + "<br><br>Le résultat est : " + result + "<br><br>");

    }
  }

  function multiplyThem(number1,number2){
    var calculation = number1 * number2;
    return calculation;
  }
  function divideThem(number1,number2){
    var calculation = number1 / number2;
    return calculation;
  }
  function modThem(number1,number2){
    var calculation = number1 % number2;
    return calculation;
  }

  document.addEventListener('DOMContentLoaded', function() {

  doMath(5,2,multiplyThem);

  doMath(10,3,divideThem);

  doMath(81,9,modThem);

  }, false);
  </script>
</head>
<body>
<h1>Faites le calcul</h1>
<div id="theResult"></div>
</body>
</html>
```

Le résultat produit par l'exécution du code du Listing 15.2 est illustré sur la Figure 15.2.

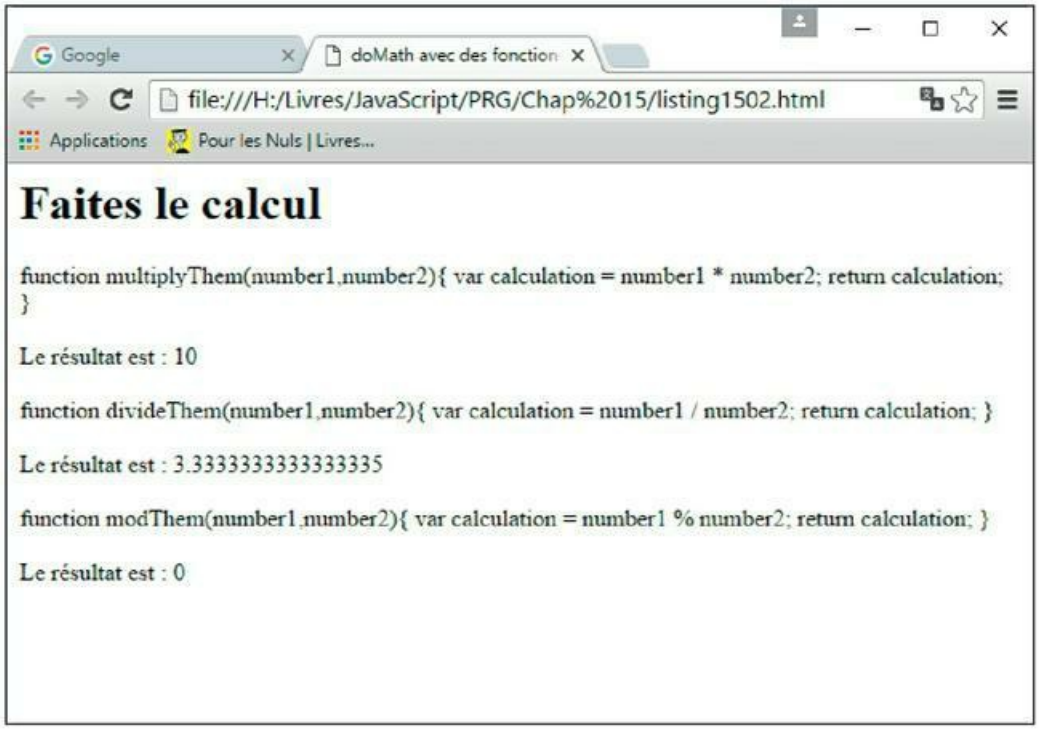

Google
doMath avec des fonction
file:///H:/Livres/JavaScript/PRG/Chap%2015/listing1502.html
Applications
Pour les Nuls | Livres...
Faites le calcul
function multiplyThem(number1,number2){ var calculation = number1 * number2; return calculation; }
Le résultat est : 10
function divideThem(number1,number2){ var calculation = number1 / number2; return calculation; }
Le résultat est : 3.3333333333333335
function modThem(number1,number2){ var calculation = number1 % number2; return calculation; }
Le résultat est : 0

FIGURE 15.2 : Faire des calculs mathématiques avec des fonctions de rappel nommées.



L'emploi de fonctions nommées pour les rappels offre deux avantages sur les fonctions anonymes :

- Le code est plus facile à lire.
- Les fonctions nommées sont à usage multiple. Autrement dit, elles peuvent être utilisées seules ou comme rappels.

Comprendre les fermetures

Une *fermeture* (ou *closure*) est une variable locale pour une fonction, dont la valeur reste « vivante » une fois l'exécution de la fonction terminée.

Prenons l'exemple du Listing 15.3. Ici, une fonction interne est définie à l'intérieur d'une autre fonction. Lorsque la fonction externe retourne une référence à la fonction interne, cette référence peut continuer à accéder à la donnée locale dans la fonction externe.

Dans le Listing 15.3, la fonction `greetVisitor` retourne une fonction créée à l'intérieur de son propre code, et qui est appelée `sayWelcome`. Remarquez que l'instruction `return` ne comporte pas de parenthèses après `sayWelcome`. En fait, vous ne voulez pas renvoyer la valeur fournie par la fonction interne, mais le code de celle-ci.

LISTING 15.3 : Créer une fonction en utilisant une fonction.

```
function greetVisitor(phrase) {
  var welcome = phrase + ". Content de vous voir !<br><br>"; // Local
variable
  var sayWelcome = function() {
    document.getElementById("greeting").innerHTML += welcome;
  }
    return sayWelcome;
}

var personalGreeting = greetVisitor("Bienvenue, ami");
personalGreeting(); // affiche "Bienvenue, ami. Content de vous voir !"
```

Ce qui est intéressant avec le Listing 15.3, c'est que ce code utilise la fonction `greetVisitor` pour créer une nouvelle fonction personnalisée appelée `personalGreeting`, cette dernière étant capable d'accéder aux variables de la fonction originale.

Normalement, lorsque l'exécution d'une fonction est terminée, ses variables locales deviennent inaccessibles. En retournant une référence à la fonction (`sayWelcome`, ici), cependant, les données de la fonction interne `greetVisitor` deviennent accessibles au monde extérieur.



Pour comprendre les fermetures, la clé consiste à intégrer la notion de portée des variables en JavaScript ainsi que la différence entre exécuter une fonction et passer une fonction par référence. En affectant la valeur de retour de `greetVisitor` à la nouvelle fonction `personalGreeting`, le programme mémorise le code de la fonction `sayWelcome`. Vous pouvez tester cela en utilisant la méthode `toString ()` :

```
personalGreeting.toString ()
```

Sur le Listing 15.4, une nouvelle fermeture a été créée en utilisant un argument différent pour la fonction `greetVisitor`. Même si l'appel à `greetVisitor` change la valeur de la variable `welcome`, le résultat de l'appel à la première fonction (`personalGreeting`) reste le même.

LISTING 15.4 : Les fermetures peuvent contenir des références secrètes aux variables de la fonction extérieure.

```
<html>
<head>
<title>Utiliser des fermetures</title>
<script>
function greetVisitor(phrase) {

  var welcome = phrase + ". Content de vous voir !<br><br>"; // Local
variable
  var sayWelcome = function() {
    document.getElementById("greeting").innerHTML += welcome;
  }
    return sayWelcome;
```

```
}
// attend jusqu'à ce que le document soit chargé
document.addEventListener('DOMContentLoaded', function() {

// crée une fonction
var personalGreeting = greetVisitor("Bienvenue, ami");
// crée une autre fonction
var anotherGreeting = greetVisitor("Salut, Ami");

// vérifie le code de la première fonction
document.getElementById("greeting").innerHTML +=
"personalGreeting.toString() <br>"
+ personalGreeting.toString() + "<br>";

// exécute la première fonction
personalGreeting(); // alerts "Hola Amiga. Great to see you!""

// vérifie le code de la seconde fonction
document.getElementById("greeting").innerHTML +=
"anotherGreeting.toString() <br>" +
anotherGreeting.toString() + "<br>";

// exécute la seconde fonction
anotherGreeting(); // alerts "Salut, Ami. Je suis content de vous voir !"

// contrôle la première fonction
personalGreeting(); // alerts "Bienvenue, ami. Content de vous voir !"

// termine la méthode addEventListener
}, false);
</script>
</head>
<body>
  <p id="greeting"></p>

</body>
</html>
```

Le résultat de l'exécution du Listing 15.4 est illustré sur la Figure 15.3.



Les fermetures ne sont pas si difficiles à comprendre dès lors que vous connaissez les concepts sous-jacents et que vous avez besoin de les utiliser. Mais ne paniquez pas si vous ne vous sentez pas tout de suite à l'aise avec elles. Il est parfaitement possible de coder en JavaScript sans utiliser de fermetures, mais, une fois que vous les avez comprises, elles peuvent être vraiment utiles et elles vous aideront à devenir un meilleur programmeur.


Google
Utiliser des fermetures
file:///H:/Livres/JavaScript/PRG/Chap%2015/listing1504.html
Applications
Pour les Nuls | Livres...
personalGreeting.toString()
function () { document.getElementById("greeting").innerHTML += welcome; }
Bienvenue, ami. Content de vous voir !
anotherGreeting.toString()
function () { document.getElementById("greeting").innerHTML += welcome; }
Salut, Ami. Content de vous voir !
Bienvenue, ami. Content de vous voir !

FIGURE 15.3 : Créer des messages personnalisés avec des fermetures.

Utiliser les fermetures

Une fermeture, c'est comme conserver une copie des variables locales d'une fonction dans l'état où elles se trouvaient avant que la fermeture n'intervienne.

Dans la programmation pour le Web, les fermetures sont souvent utilisées pour éliminer tout effort de duplication dans une application, ou encore pour conserver en l'état des valeurs qui ont besoin d'être réutilisées par la suite de manière à ce que le programme n'ait pas besoin de les recalculer à chaque fois qu'il doit les retrouver.

Un autre emploi des fermetures consiste à créer des versions personnalisées de fonctions pour des usages spécifiques.

Sur le Listing 15.5, des fermetures sont utilisées afin de créer des fonctions affichant des messages d'erreurs spécifiques selon les problèmes qui peuvent survenir dans le programme. Tous ces messages d'erreur sont produits en utilisant la même fonction (c'est ce que l'on appelle parfois une *fabrique de fonction*).

LISTING 15.5 : Utiliser une fonction pour créer d'autres fonctions.

```
<html>
  <head>
    <title>La fabrique de fonction</title>
   <script>

    function createMessageAlert(theMessage){
      return function() {
        alert (theMessage);
      }
    }

    var badEmailError = createMessageAlert("Adresse email inconnue !");
    var wrongPasswordError = createMessageAlert("Mot de passe invalide
!");

    window.addEventListener('load', loader, false);
    function loader(){

      document.login.yourEmail.addEventListener('change',badEmailError);

document.login.yourPassword.addEventListener('change',wrongPasswordError);



    }

    </script>
  </head>
  <body>
    <form name="login" id="loginform">
      <p>
        <label>Entrez votre adresse email :
        <input type="text" name="yourEmail">
        </label>
      </p>
      <p>
        <label>Entrez votre mot de passe :
        <input type="text" name="yourPassword">
        </label>
      </p>
      <button>Soumettre</button>
  </body>
</html>
```

Pour comprendre le comportement du code du Listing 15.5, le point clé réside dans la « fabrique » de fonction :

```
function createMessageAlert(theMessage){
      return function() {
        alert (theMessage);
      }
    }
```

Pour utiliser ce code, vous affectez sa valeur de retour à une variable, comme dans l'instruction suivante :

```
var badEmailError = createMessageAlert ("Adresse email inconnue ! ") ;
```

L'instruction précédente crée une fermeture qui peut être utilisée partout ailleurs dans le programme, simplement en exécutant `badEmailError` en tant que fonction, comme dans le gestionnaire d'événement suivant :

```
document.login.yourEmail.addEventListener('change',badEmailError);
```

Chapitre 16

Toujours plus loin avec Ajax et JSON

DANS CE CHAPITRE :

- **Lire et écrire avec JSON**
- **Comprendre Ajax**
- **Utiliser Ajax**

« Le Web ne fait pas que connecter des machines. Il connecte des gens. »

Tim Berners-Lee

Ajax est une technique qui permet de rendre les pages plus dynamiques en envoyant et recevant les données en arrière-plan pendant que l'utilisateur interagit avec elles. JSON est devenu le format de données standard utilisé par les applications Ajax. Dans ce chapitre, vous allez apprendre comment l'emploi des techniques Ajax peut rendre votre site plus étincelant !



N'oubliez pas de visiter le site Web compagnon de ce livre pour y télécharger les exemples de ce chapitre !

Travailler en coulisses avec Ajax

Ajax est un acronyme qui signifie *Asynchronous JavaScript + XML*. Ce terme est utilisé pour décrire une méthode d'utilisation de JavaScript, du modèle DOM, de HTML et de l'objet `XMLHttpRequest` pour rafraîchir des parties d'une page Web avec des données dynamiques sans avoir à recharger la page entière. Ajax a d'abord été implémenté à une large échelle par Google pour sa messagerie Gmail en 2004, et son nom définitif lui a été donné par James Garret en 2005.

Le modèle DOM de HTML change la page de manière dynamique. L'importante innovation apportée par Ajax a été l'utilisation de l'objet `XMLHttpRequest` pour retrouver des données sur le serveur de façon asynchrone (en arrière-plan, si vous préférez), sans blocage de l'exécution du reste du code JavaScript dans la page Web.

Bien qu'Ajax reposait à l'origine sur des données au format XML (pour *Extensible Markup Language*, ou langage de balisage extensible), il est plus courant aujourd'hui pour les applications Ajax de faire appel à un format appelé JSON (pour *JavaScript Object Notation*). Bien entendu, dans ce cas, le *X* final devrait plutôt être remplacé par un *J* pour donner Ajaj, mais il faut reconnaître que cette appellation serait moins poétique.

Exemples Ajax

Lorsque les développeurs Web ont commencé à utiliser Ajax, celui-ci est devenu l'un des marqueurs de ce qu'on a appelé le Web 2.0. Avant Ajax, la méthode la plus courante pour montrer des données dynamiques consistait à télécharger une nouvelle page depuis le serveur.

Prenons l'exemple du site illustré sur la Figure 16.1. Pour naviguer entre les catégories ou les résultats des recherches, vous pouvez cliquer sur des liens qui provoquent le rafraîchissement de l'ensemble de la page. Même si elle reste toujours très courante, la technique consistant à recharger toute la page alors que seule une partie de

son contenu a changé est inutilement lente, ce qui peut de surcroît laisser une impression un peu désagréable à l'utilisateur.

Comparons maintenant le site précédent avec celui des amis de Google, Google Plus, illustré sur la Figure 16.2. Google Plus utilise Ajax pour charger de nouveaux contenus dans les « pavés » de la page, tandis que la barre de navigation reste statique.

En plus de rendre la navigation plus fluide, Ajax est également excellent pour créer des éléments « vivants » dans une page Web. Avant Ajax, si vous vouliez afficher des données dynamiques, comme un graphique en temps réel, ou encore une vue actualisée de la boîte de réception d'une messagerie, vous deviez soit utiliser un plug-in (comme Adobe Flash), ou provoquer périodiquement un rafraîchissement de la page.

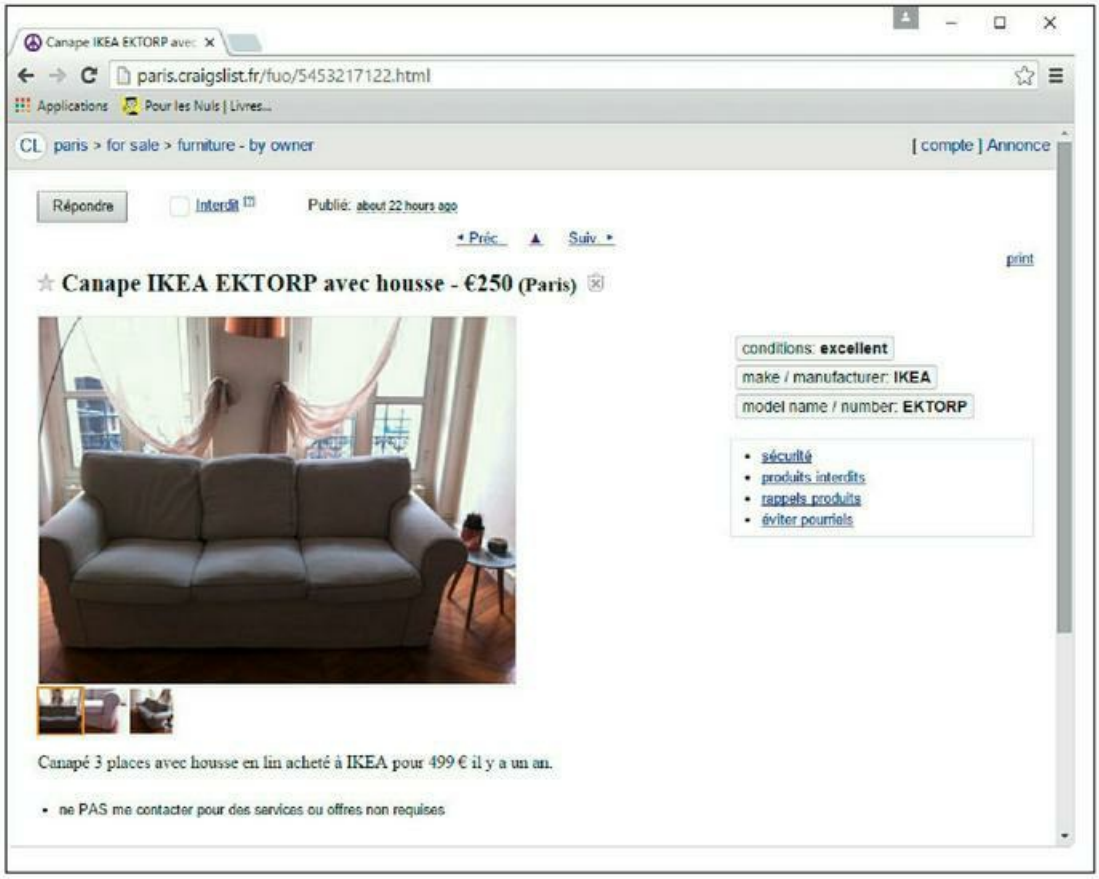

Canape IKEA EKTORP avec
paris.craigslist.fr/fuo/5453217122.html
Applications
Pour les Nuls | Livres...
CL paris > for sale > furniture - by owner
[compte] Annonce
Répondre
Interdit
Publié: about 22 hours ago
« Préc.
Suiv »
print
Canape IKEA EKTORP avec housse - €250 (Paris)
conditions: excellent
make / manufacturer: IKEA
model name / number: EKTORP
sécurité
produits interdits
rappels produits
éviter pourriels
Canapé 3 places avec housse en lin acheté à IKEA pour 499 € il y a un an.
ne PAS me contacter pour des services ou offres non requises

FIGURE 16.1 : Certains sites se satisfont du Web 1.0. Et vous ?

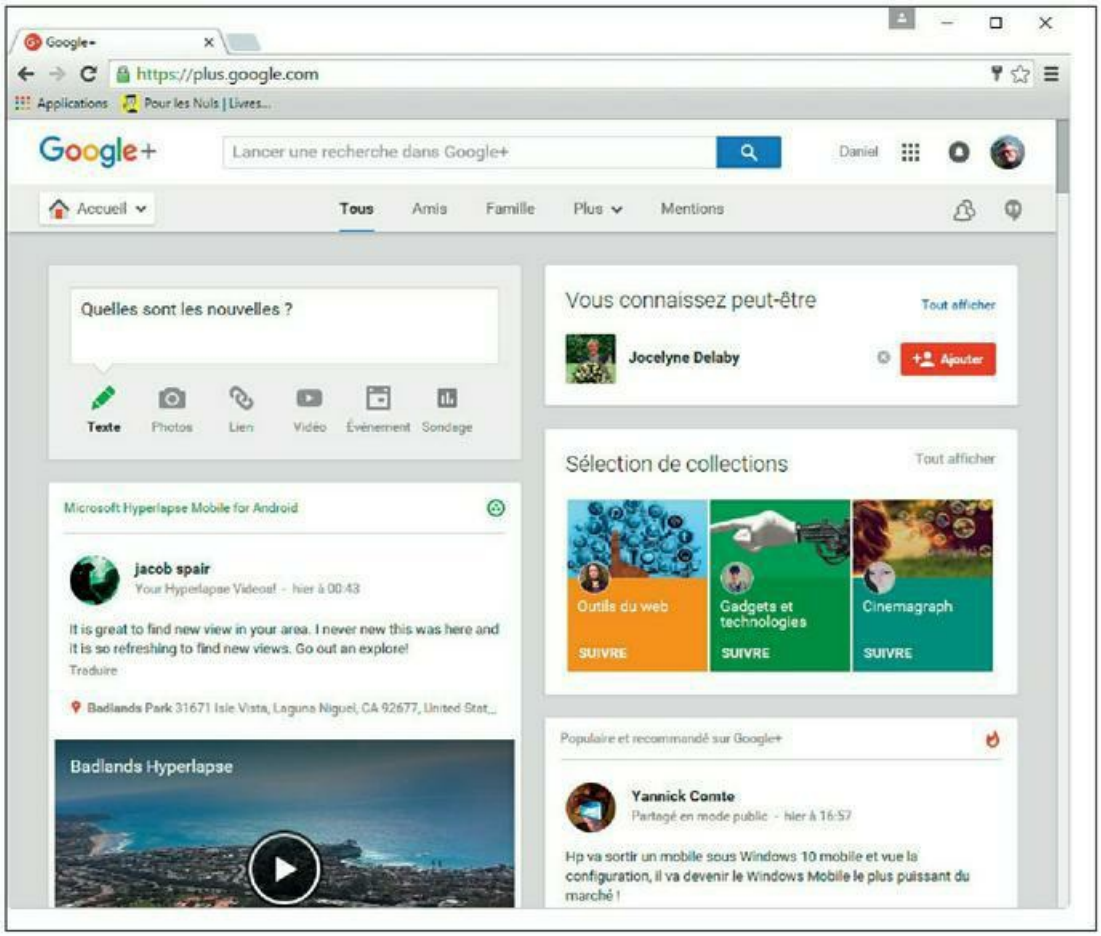


FIGURE 16.2 : Google Plus utilise Ajax pour fournir une expérience utilisateur moderne.

Avec Ajax, il est possible d'actualiser périodiquement les données grâce à un processus asynchrone qui s'exécute en arrière-plan, et qui ne met à jour que les éléments de la page qui ont besoin d'être modifiés.

Le Chapitre 10 explique comment actualiser le contenu HTML et CSS d'une page Web en utilisant les méthodes et les propriétés du modèle DOM. Ajax repose sur ces mêmes techniques.

Le site Wundermap (`https://www.wunderground.com/wundermap/`), illustré sur la Figure 16.3, montre une carte météo qui change constamment. Les données de cette carte sont récupérées sur des serveurs distants en utilisant Ajax.

Voir Ajax en action

Sur la Figure 16.3, les outils de développement de Chrome ont été ouverts en activant l'onglet Network (réseau). Celui-ci montre toutes les activités réseau concernant la page Web courante. Lorsqu'une page est chargée, cela concerne les requêtes et les téléchargements du code HTML, CSS et JavaScript ainsi que des images. Une fois ce chargement terminé, l'onglet Network affiche aussi les requêtes et les réponses HTTP asynchrones qui rendent Ajax possible.

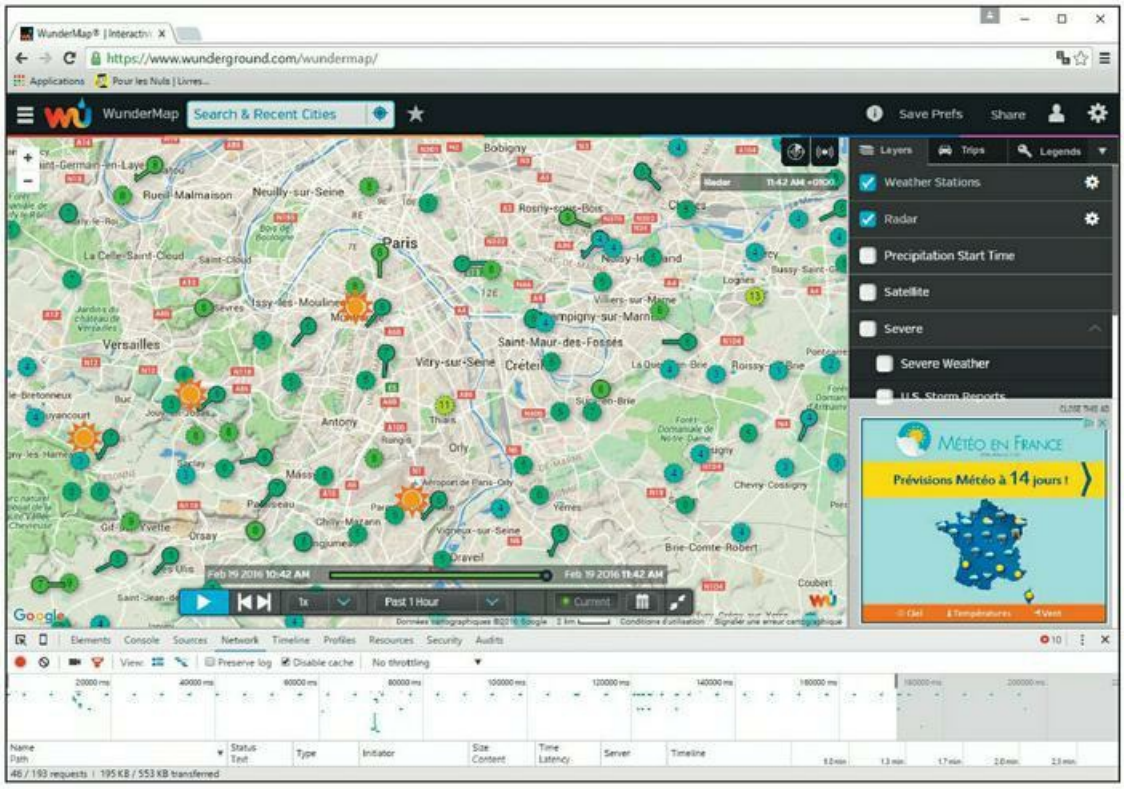


FIGURE 16.3 : Wundermap utilise Ajax pour afficher des données météo très régulièrement actualisées.

Pour mieux voir cela en action, suivez ces étapes :

1. **Ouvrez votre navigateur Chrome et naviguez vers le site https://www.wunderground.com/wundermap/.**
2. **Ouvrez les outils de développement de Chrome en utilisant son menu, ou en appuyant sur Command + Option + I (Mac) ou Ctrl + Maj + I (Windows).**
3. **Ouvrez l'onglet Network des outils de développement.**

   À ce stade, votre fenêtre devrait ressembler à celle de la Figure 16.3. Autrement dit, elle affiche une ligne de temps qui montre la charge de travail de la page Web. Pour révéler davantage d'informations, faites glisser vers le haut la ligne de partage entre la page et les outils de développement. Ne vous souciez pas du fait que le contenu de la page devient largement masqué : c'est bien entendu ce qu'affichent les outils de développement qui nous intéresse ici. Vous devriez alors voir quelque chose qui ressemble plus à l'illustration de la Figure 16.4.

   Remarquez que de nouveaux éléments apparaissent périodiquement sous l'onglet Network. Il s'agit des requêtes et des réponses Ajax. Certaines sont des images renvoyées par le serveur, d'autres sont des données utilisées par le code JavaScript côté client.

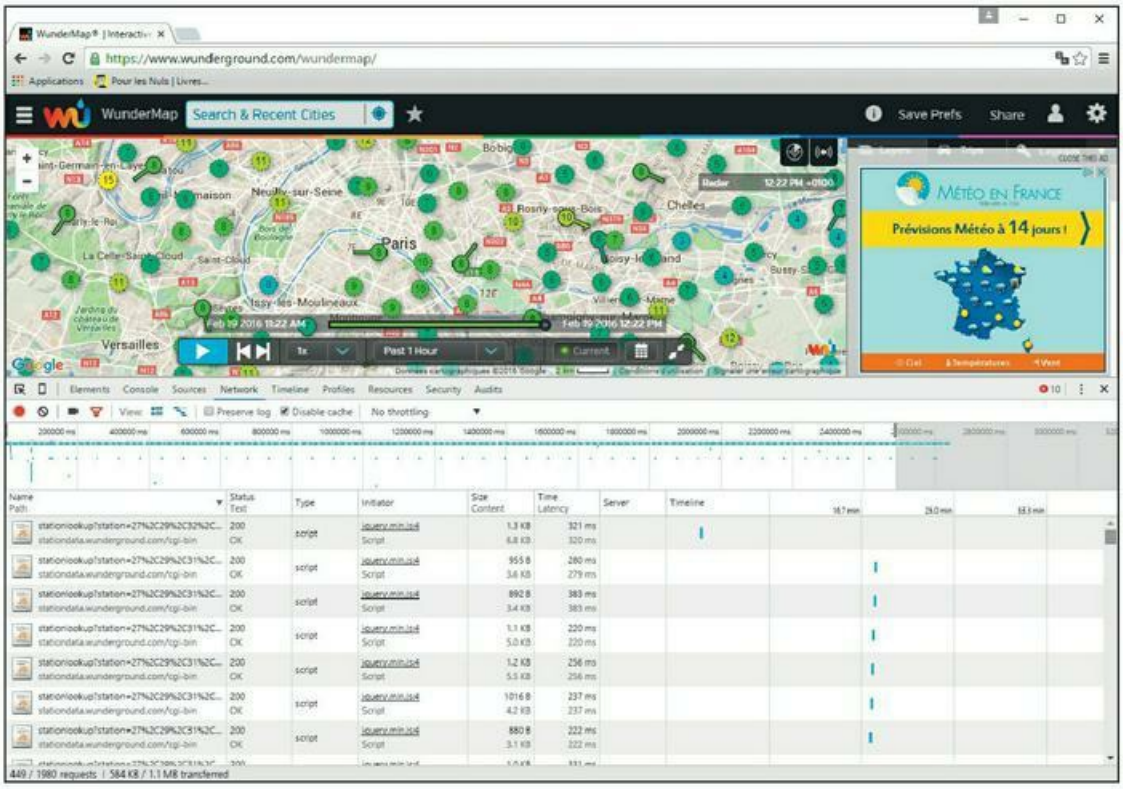


FIGURE 16.4 : L'onglet Network des outils de développement de Chrome est développé.

4. **Cliquez sur une des lignes de la colonne Name de l'onglet Network.**

   Des données spécifiques à cet élément sont affichées (voir la Figure 16.5).

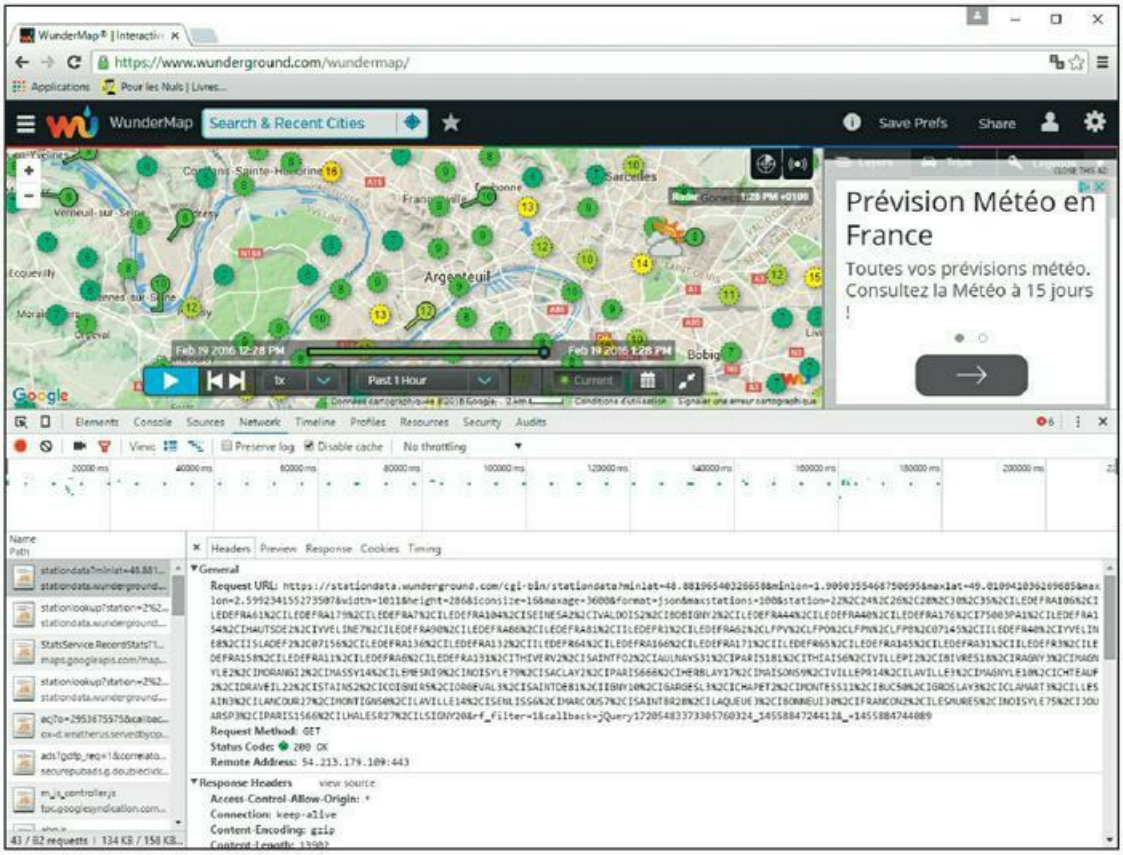


FIGURE 16.5 : Afficher des informations supplémentaires pour un enregistrement particulier de l'onglet Network.

5. **Cliquez sur les onglets (Headers, Preview, Response, etc.) dans la vue détaillée et examinez les données.**

   Le premier onglet, Headers, affiche la requête HTTP qui a été envoyée au serveur distant. Regardez en particulier ce qu'indique Request URL. Il s'agit d'une adresse Web standard qui passe des données à un serveur distant.

6. **Sélectionnez et copiez la valeur de la section Request URL dans l'élément que vous inspectez.**

7. **Ouvrez un nouvel onglet, et collez dans la barre d'adresse la valeur de Request URL que vous venez de copier.**

   Une page contenant des données ou une image va s'ouvrir (voir la Figure 16.6).

8. **Comparez ce que vous venez d'obtenir avec ce qu'affiche l'onglet Response des outils de développement de Chrome.**

   Les résultats devraient être comparables, même s'ils ne sont pas identiques du fait qu'ils n'ont pas été obtenus simultanément.

Comme vous pouvez le constater, il n'y a aucune magie dans Ajax. Le code JavaScript de la page Web envoie simplement des requêtes à un serveur distant, et il reçoit en retour des données. Tout ce qui se passe dans les coulisses peut être scruté à l'aide des outils de développement de Chrome (ou des outils semblables proposés par d'autres navigateurs, comme Firefox).

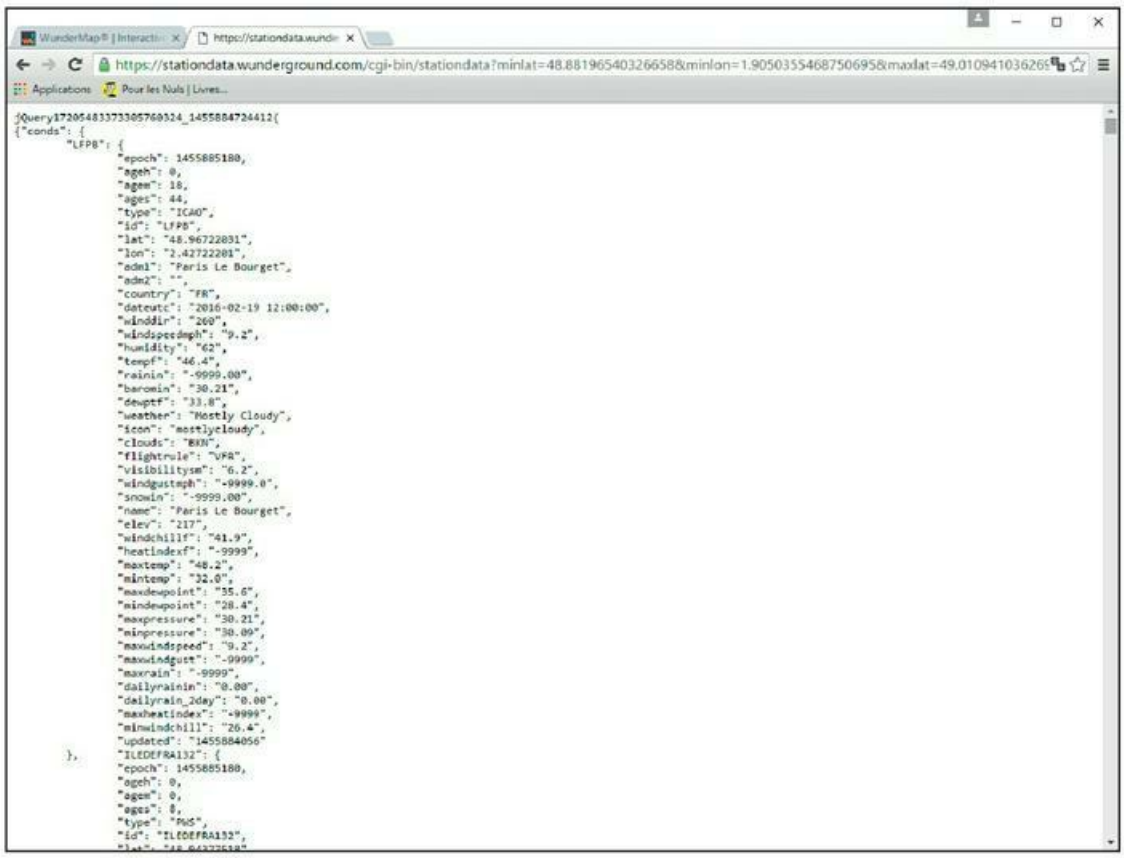


FIGURE 16.6 : Un résultat affiché en copiant l'adresse http de Request URL.

Utiliser l'objet XMLHttpRequest

L'objet `XMLHttpRequest` fournit un moyen pour les navigateurs Web d'envoyer une requête au serveur pour obtenir en retour des données sans avoir à rafraîchir la page.

L'objet `XMLHttpRequest` a été créé et implémenté en premier par Microsoft dans son navigateur Internet Explorer. C'est devenu depuis un standard du Web, et il a été adapté par tous les navigateurs Web modernes.

Vous pouvez utiliser les méthodes et les propriétés de l'objet `XMLHttpRequest` pour récupérer des données depuis un serveur distant ou sur votre serveur local. En dépit de son nom, l'objet `XMLHttpRequest` peut obtenir d'autres types de données que du XML, et il est capable d'utiliser pour cela différents protocoles en plus de HTTP.

Le Listing 16.1 montre comment vous pouvez utiliser `XMLHttpRequest` pour charger le contenu d'un document de texte externe (contenant en l'occurrence une ligne de code HTML) dans la page Web HTML courante.

LISTING 16.1 : UTILISER XMLHttpRequest pour charger des données externes.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
    <title>Charger des données externes</title>
    <script>
    window.addEventListener('load',init,false);
    function init(e){

document.getElementById('myButton').addEventListener('click',documentLoader,

        false);
    }

    function reqListener () {
      console.log(this.responseText);
      document.getElementById('content').innerHTML = this.responseText;

    }

    function documentLoader(){
      var oReq = new XMLHttpRequest();
      oReq.onload = reqListener;
      oReq.open("get", "loadme.txt", true);
      oReq.send();
    }
    </script>
  </head>
  <body>
  <form id="myForm">
    <button id="myButton" type="button">Cliquer pour charger</button>
  </form>
  <div id="content"></div>
  </body>
</html>
```

La première ligne de code à l'intérieur de la fonction crée le nouvel objet `XMLHttpRequest` et lui donne comme nom `oReq` :

```
var oReq = new XMLHttpRequest () ;
```

Les méthodes et propriétés de l'objet `XMLHttpRequest` sont maintenant accessibles *via* l'objet `oReq`.

La seconde ligne assigne une fonction, reqListener, à l'événement `onload` de l'objet `oReq`. Le but de cette instruction est de forcer l'appel à `reqListener` lorsque `oReq` charge un document :

```
oReq.onload = reqListener ;
```

La troisième ligne utilise la méthode `open` pour créer une requête :

```
oReq.open ("get", "loadme.txt", true) ;
```

Dans ce cas, la fonction utilise la méthode HTTP `get` pour charger le fichier appelé `loadme.txt`. Le dernier paramètre est l'argument `async`. Il spécifie le mode synchrone ou asynchrone de la requête. Si sa valeur est `false`, la méthode `send` ne retournera rien jusqu'à ce que la requête soit totalement traitée. Si sa valeur est `true`, des notifications sur l'état d'avancement de la requête seront fournies *via* un gestionnaire d'événement. Du fait que le gestionnaire est configuré ici pour surveiller l'événement `load`, une requête asynchrone est ce qui nous intéresse.



Il est peu probable que vous puissiez vous trouver dans une situation où vous voudriez définir l'argument `async` avec la valeur `false`. En fait, certains navigateurs ont commencé à ignorer cet argument s'il vaut `false`, donc à le traiter comme valant `true` dans tous les cas, et ce afin d'éliminer le mauvais effet que peuvent avoir sur l'expérience utilisateur les requêtes synchrones.

La dernière ligne de la fonction `documentLoader` transmet la requête créée avec la méthode `open` :

```
oReq.send ;
```



La méthode `open` récupère la dernière version du fichier demandé. Les applications réellement dynamiques utilisent souvent des boucles pour demander de manière répétée les données disponibles les plus récentes à un serveur utilisant Ajax.

Travailler avec la contrainte same-origin policy

Si vous exécutez le code du Listing 16.1 dans un navigateur Web, vous n'allez pas obtenir le résultat que vous escomptiez. Si vous ouvrez par exemple la console dans les outils de développement de Chrome, vous devriez voir (au moins) un message d'erreur semblable à celui qui est illustré sur la Figure 16.7.

Le problème, ici, tient à ce qui est appelé *same-origin policy* (règle de même origine). Pour éviter que des pages Web puissent, à l'insu de l'utilisateur, télécharger par l'intermédiaire de `XMLHttpRequest` du code qui pourrait être dommageable (virus ou autre), les navigateurs renvoient par défaut une erreur lorsqu'un script essaie de charger une URL qui n'a pas la même origine que l'adresse courante. Si vous ouvrez par exemple une page Web provenant du site `www.site1.com`, et qu'un script tente de retrouver des données à l'adresse `www.site2.com`, le navigateur réagira par une erreur semblable à celle qui apparaît sur la Figure 16.7.

Cette règle vaut aussi pour les fichiers qui sont enregistrés sur votre ordinateur local. Si ce n'était pas le cas, `XMLHttpRequest` pourrait être utilisé pour compromettre la sécurité de votre système (et donc la vôtre).

Dans le cas des exemples de ce livre, il n'y a absolument aucune raison de s'inquiéter à ce sujet. Cependant, pour que les exemples de ce chapitre fonctionnent correctement sur votre ordinateur, il faut bien qu'il existe un moyen de contourner la règle *same-origin policy*.

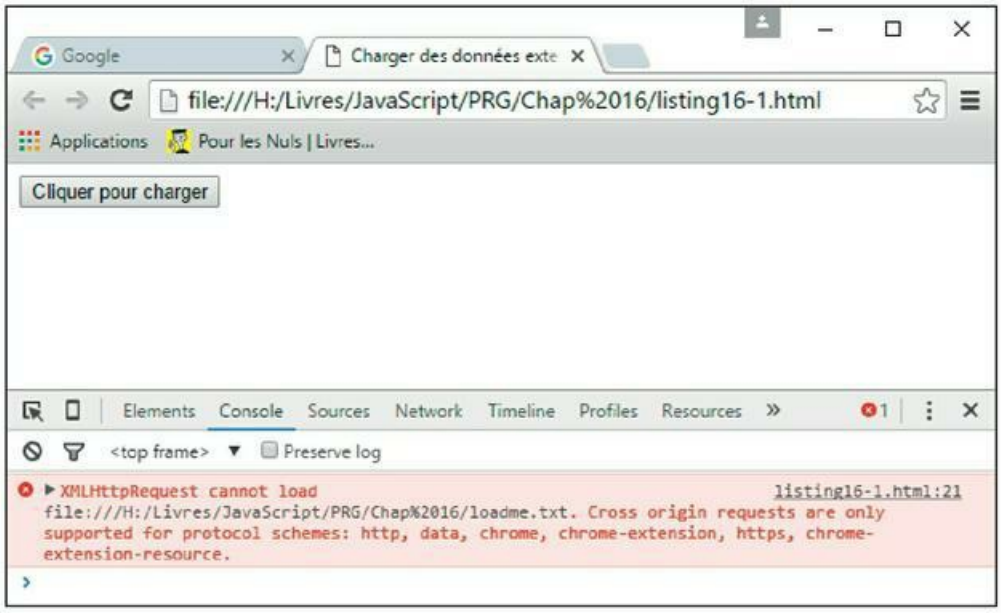


FIGURE 16.7 : Utiliser XMLHttpRequest sur un fichier local provoque une erreur.

La première méthode consiste à placer le fichier HTML qui contient la fonction `documentLoader` et le fichier de texte ensemble sur le même serveur Web.

La seconde méthode revient à désactiver temporairement les restrictions same-origin policy dans votre navigateur.



Ces instructions vous permettent de tester vos propres fichiers (dans le cas où vous n'avez pas installé un serveur Web sur votre machine) uniquement sur votre propre ordinateur. Ne surfez pas sur le Web avec les restrictions same-origin policy désactivées ! Vous pourriez exposer votre ordinateur à être infecté par du code malveillant.

Pour désactiver les restrictions same-origin policy sous Mac OS :

1. **Refermez votre navigateur Chrome.**
2. **Ouvrez l'application Terminal et relancez Chrome en utilisant la commande suivante :**

```
/Applications/Google\ Chrome.app/Contents/MacOS/Google\ Chrome --disable-
web-
security
```

Pour désactiver les restrictions same-origin policy sous Windows :

1. **Refermez votre navigateur Chrome.**
2. **Ouvrez l'invite de commande et naviguez vers le dossier dans lequel vous avez installé Chrome (normalement,** `c : \Program Files (x86)\ Google\Chrome\Application`**).**
3. **Tapez la commande suivante pour relancer le navigateur :**

```
Chrome.exe --disable-web-security
```

Vous pouvez maintenant exécuter localement des fichiers contenant des requêtes Ajax, et ce jusqu'à ce que vous refermiez la fenêtre du navigateur. Lors de la prochaine ouverture, les restrictions de sécurité seront automatiquement réactivées.

La Figure 16.8 montre le résultat produit dans ce cas par l'exécution du Listing 16.1 lorsque l'utilisateur clique sur le bouton Cliquer pour charger.

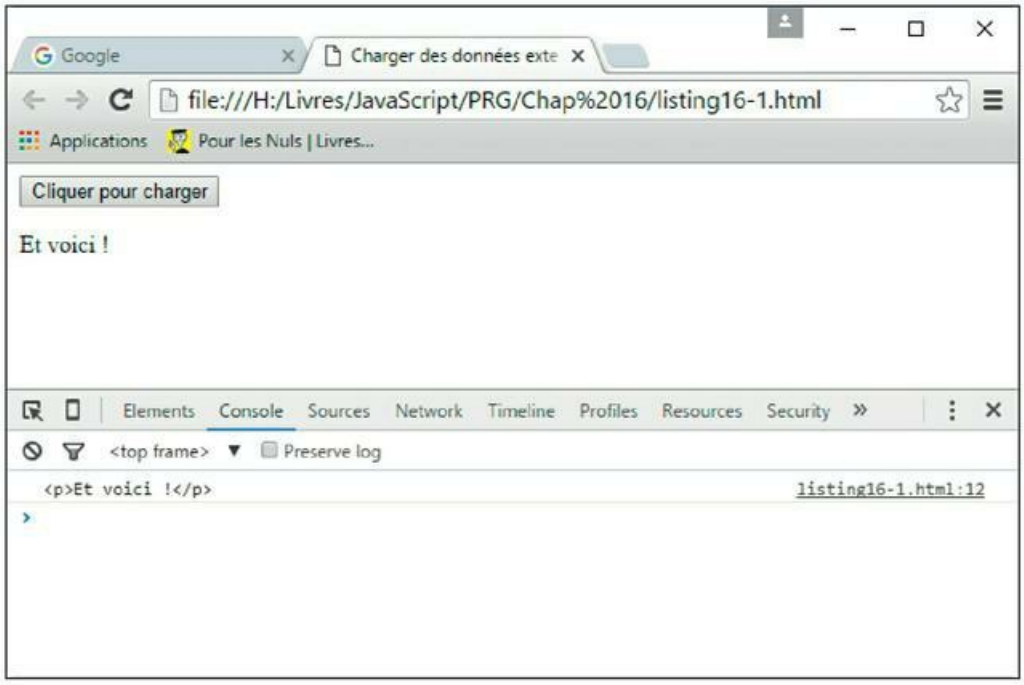


FIGURE 16.8 : Le Listing 16.1 est exécuté dans un navigateur où les restrictions same-origin policy sont désactivées.

Utiliser CORS pour les requêtes Ajax

Il est assez courant qu'une application Web ait besoin d'adresser des requêtes à des serveurs différents pour obtenir les données dont elle a besoin. Par exemple, Google fournit ses données géographiques gratuitement à des applications tierces.

Pour que les transactions entre serveurs soient sécurisées, il faut disposer de mécanismes pour que les navigateurs et les serveurs soient capables de faire fi de leurs différences afin de pouvoir collaborer en toute confiance.

Jusqu'ici, la meilleure méthode permettant d'autoriser ou d'interdire le partage de ressources entre serveurs est le standard appelé CORS (pour *Cross-Origin Resource Sharing*, ou encore croisement de ressources – ou de domaine).

Pour voir CORS en action, jetez un coup d'œil à l'onglet Network des outils de développement de Chrome tout en naviguant sur le site Wundermap. Pour cela, cliquez sur un des liens qui indique `http://stationdata.wunderground.com/cgi-bin/stationlookup`.

Vous devriez voir sous l'onglet Headers le texte suivant dans l'en-tête http:

```
Access-Control-Allow-Origin : *
```

Il s'agit de l'en-tête de la réponse CORS pour laquelle ce serveur particulier est configuré. L'astérisque après le deux-points indique que ce serveur peut accepter des requêtes de n'importe quelle origine. Si le propriétaire du site `www.wunderground.com` voulait restreindre l'accès aux données provenant de certains serveurs spécifiques ou d'utilisateurs authentifiés, il utiliserait aussi CORS.

Mettre des objets en mouvement avec JSON

Sur le Listing 16.1, vous avez utilisé Ajax pour ouvrir et afficher un document de texte contenant une simple ligne de code HTML. Un autre usage courant d'Ajax consiste à demander et à recevoir des données pour qu'elles soient traitées par le navigateur.

Par exemple, nos amis américains disposent d'un site, `gasbuddy.com`, qui leur permet d'utiliser une carte fournie par Google pour afficher des prix du gaz à tel ou tel endroit, comme l'illustre la Figure 16.9.

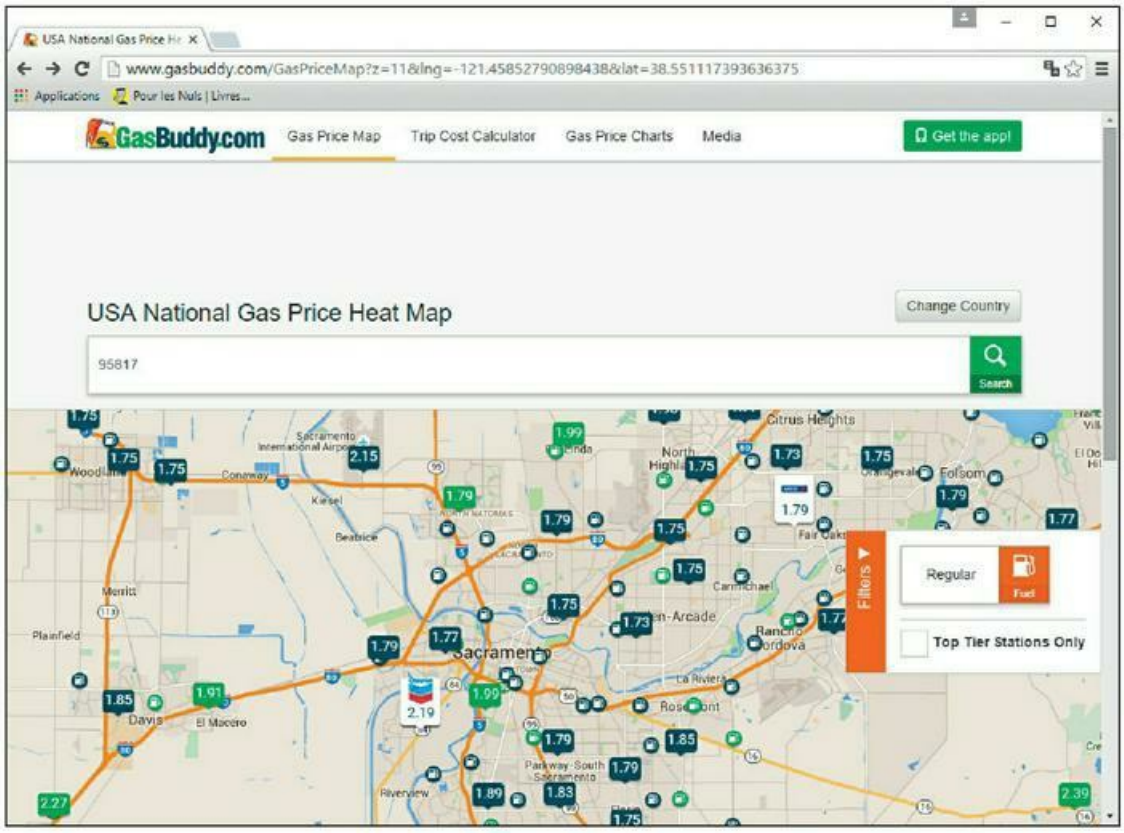


FIGURE 16.9 : Le site gasbuddy.com utilise Ajax pour afficher les prix du gaz sur une carte.

Si vous examinez ce qu'indique l'onglet Network dans `gasbuddy.com`, vous pourrez constater que certaines requêtes ont des réponses dont le code ressemble à celui du Listing 16.2.

LISTING 16.2 : Une partie d'une réponse à une requête Ajax sur le site gasbuddy.com.

```
([{id:"tuwtvtuvvvv",base:[351289344,822599680],zrange:[
11,11],layer:"m@288429816",features:[{id:"172
438574634854 76481",a:[0,0],bb:[-8,-8,7,7,-47-
,7,48,22,-41,19,41,34],c:"{1:{title:\"Folsom
Lake State Recreation Area\"},4:{type:1}}"}]},
{id:"tuwtvtuvvvw",zrange:[11,11],layer:"m@2884
29816"},{id:"tuwtvtuvvwv",base:[351506432,8242
91328],zrange:[11,11],layer:"m@288429816",feat
ures:[{id:"87485585183532 72790",a:[0,0],bb:[-
8,-8,7,7,-41,7,41,22],c:"{1:{title:\"Deer Creek
Hills\"},4:{type:1}}"}]},{id:"tuwtvtuvvww",zran
ge:[11,11],layer:"m@288429816"}])
```

Si vous prenez une partie de ce code et que vous le reformatez, vous obtiendrez quelque chose qui ressemble au Listing 16.3, qui présente un aspect un peu plus familier.

LISTING 6.3 : Les données fournies en réponse à gasbuddy.com, reformatées.

```
{id:"tuwtvtuvvvv",
base:[351289344,822599680],
zrange:[11,11],
layer:"m@288429816",
features:[{
id:"17243857463485476481",
a:[0,0],
bb:[-8,-8,7,7,-47,7,48,22,-41,19,41,34],
c:"{
1:{title:\"Folsom Lake State Recreation Area\"},
4:{type:1}
}"}
]}
}
```

En y regardant de près, vous pourrez remarquer que ces données ressemblent furieusement au format `nom : valeur` d'un objet littéral de JavaScript (voyez à ce sujet le Chapitre 8).

La raison principale pour laquelle JSON est si facile à utiliser est qu'il propose un format avec lequel JavaScript est déjà capable de travailler. Aucune conversion n'est donc nécessaire. Par exemple, le Listing 16.4 montre un fichier JSON contenant des informations sur ce livre.

LISTING 16.4 : Données JSON décrivant le codage de ce livre.

```
{
  "book_title": "Coder avec JavaScript pour les Nuls",
  "book_author": "Chris Minnick et Eva Holland",
  "summary": "Tout ce que les débutants doivent savoir pour commencer à
coder avec
        JavaScript  !",
  "isbn":"9781119056072"
}
```

Le Listing 16.5 illustre la manière dont ces données pourraient être chargées dans une page Web en utilisant JavaScript, puis être utilisées pour les afficher en HTML.

LISTING 16.5 : Afficher des données JSON avec JavaScript.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
    <title>Afficher des données JSON</title>
    <script>
```

```
    window.addEventListener('load',init,false);
    function init(e){

document.getElementById('myButton').addEventListener('click',documentLoader,

        false);
    }

    function reqListener () {
      // convertit la chaîne du fichier en objet avec  JSON.parse
      var obj = JSON.parse(this.responseText);

      // affiche les données comme un objet quelconque
      document.getElementById('book_title').innerHTML = obj.book_title;
      document.getElementById('book_author').innerHTML = obj.book_author;
      document.getElementById('summary').innerHTML = obj.summary;

    }

    function documentLoader(){
      var oReq = new XMLHttpRequest();
      oReq.onload = reqListener;
      oReq.open("get", "listing16-4.json", true);
      oReq.send();
    }
    </script>
  </head>
  <body>
  <form id="myForm">
    <button id="myButton" type="button">Cliquez pour charger</button>
  </form>
  <h1>Titre du livre</h1>
  <div id="book_title"></div>
  <h2>Auteurs</h2>
  <div id="book_author"></div>
  <h2>Résumé</h2>
  <div id="summary"></div>
  </body>
</html>
```

Le point clé, pour afficher des données JSON qui doivent être incorporées dans un document JavaScript en provenance d'une source extérieure, consiste à les convertir d'une chaîne en un objet *via* la méthode `JSON.parse`. Une fois que vous avez fait cela, vous pouvez accéder aux valeurs contenues dans le document JSON en utilisant la notation point ou entre crochets droits, comme vous le feriez pour accéder aux propriétés de n'importe quel objet JavaScript.

La Figure 16.10 illustre le résultat de l'exécution du Listing 16.5 dans un navigateur Web. Vous devez pour cela lancer le navigateur en mode non protégé (pour pouvoir accéder au fichier local) et cliquer sur le bouton principal.

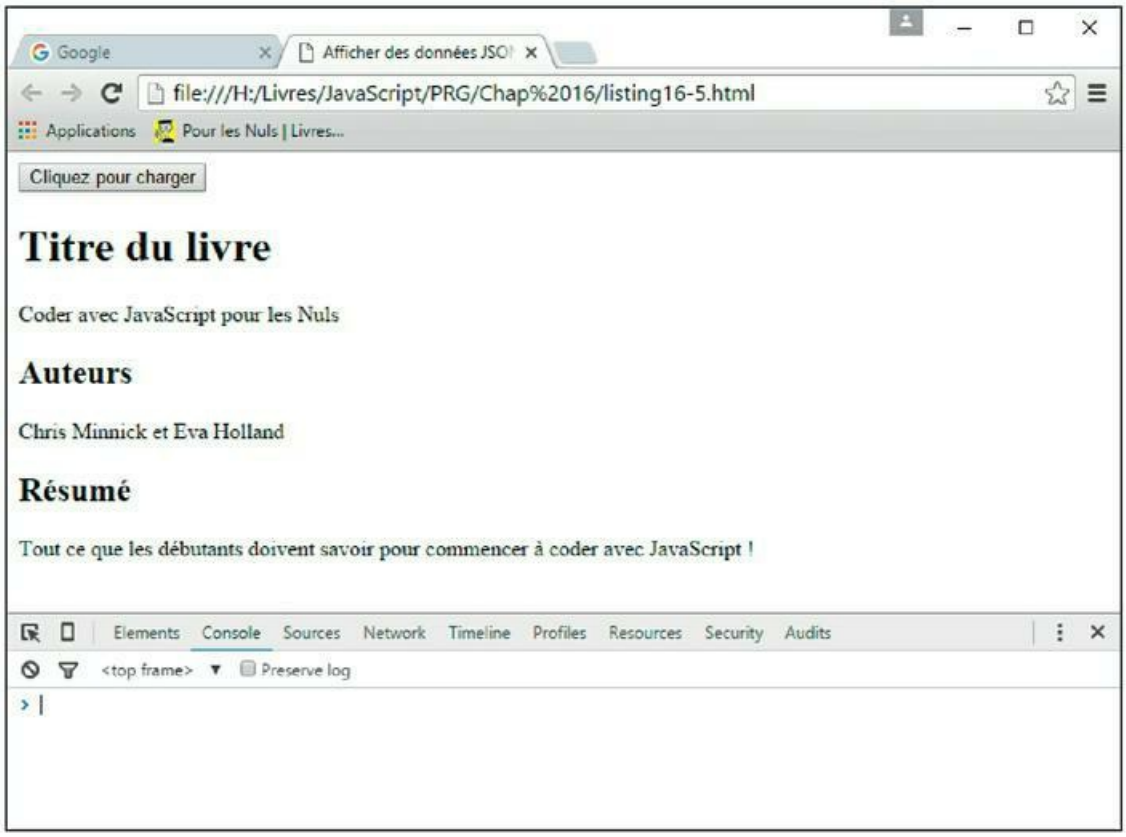


FIGURE 16.10 : Afficher des données JSON dans une page HTML.

JavaScript et HTML5

DANS CETTE PARTIE...

Comprendre comment utiliser les API de HTML5 pour accéder à de multiples fonctionnalités sur un ordinateur ou un dispositif portable

Découvrir les bases de jQuery pour accéder et simplifier le développement d'applications JavaScript

Chapitre 17

HTML5 et ses API

DANS CE CHAPITRE :

- **Découvrir les API**
- **Utiliser la géolocalisation**
- **Accéder au multimédia**

« Le langage est un virus venu de l'espace. »

William S. Burroughs

Les API de HTML5 vous donnent accès à de multiples fonctionnalités, aussi bien sur ordinateur que sur des dispositifs mobiles. Dans ce chapitre, vous allez découvrir comment vous servir de ces API, quelles sont les méthodes et les techniques standard qu'elles utilisent, ainsi que des exemples de code qui vous permettront de pénétrer dans cet univers excitant.



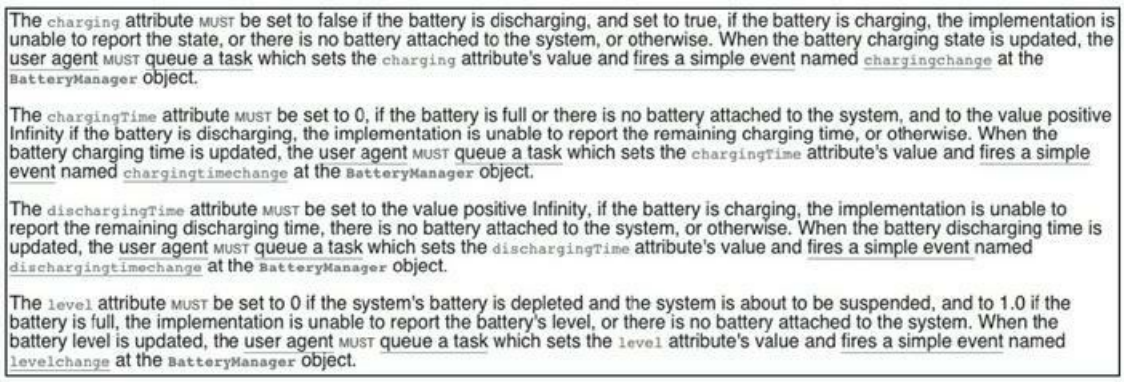
N'oubliez pas de visiter le site Web compagnon de ce livre pour y télécharger les exemples de ce chapitre !

Comprendre comment les API fonctionnent

Une API (pour *Application Programming Interface*, ou interface de programmation d'application) est un ensemble de routines et de standards logiciels qui permettent au programmeur d'accéder aux fonctionnalités d'une application. Les API sont le moyen grâce auquel un programme informatique offre à un autre programme la possibilité d'interagir avec lui. Ainsi, lorsqu'un navigateur Web permet à JavaScript d'interagir avec lui, il le fait en utilisant des API.

Par exemple, les API Battery Status définies par le consortium W3C décrivent la manière dont les navigateurs devraient se comporter devant ce qui se passe avec la batterie d'un certain dispositif, comme un smartphone ou une tablette. D'autres programmes (comme des scripts JavaScript à l'intérieur de pages Web) peuvent alors accéder à cette API pour déterminer quel est l'état de charge de la batterie de l'appareil, par exemple si sa capacité est faible, si elle est en charge, quel est le temps écoulé depuis que l'appareil a été allumé, et ainsi de suite. Ces informations sont ensuite utilisables dans votre programme.

Le langage d'une API est extrêmement précis, et il décrit exactement la manière dont cette API devrait être implémentée dans les navigateurs Web. Ainsi, la Figure 17.1 montre un extrait de la version la plus récente de l'API Battery Status.


The charging attribute MUST be set to false if the battery is discharging, and set to true, if the battery is charging, the implementation is unable to report the state, or there is no battery attached to the system, or otherwise. When the battery charging state is updated, the user agent MUST queue a task which sets the charging attribute's value and fires a simple event named chargingchange at the BatteryManager object.

The chargingTime attribute MUST be set to 0, if the battery is full or there is no battery attached to the system, and to the value positive Infinity if the battery is discharging, the implementation is unable to report the remaining charging time, or otherwise. When the battery charging time is updated, the user agent MUST queue a task which sets the chargingTime attribute's value and fires a simple event named chargingtimechange at the BatteryManager object.

The dischargingTime attribute MUST be set to the value positive Infinity, if the battery is charging, the implementation is unable to report the remaining discharging time, there is no battery attached to the system, or otherwise. When the battery discharging time is updated, the user agent MUST queue a task which sets the dischargingTime attribute's value and fires a simple event named dischargingtimechange at the BatteryManager object.

The level attribute MUST be set to 0 if the system's battery is depleted and the system is about to be suspended, and to 1.0 if the battery is full, the implementation is unable to report the battery's level, or there is no battery attached to the system. When the battery level is updated, the user agent MUST queue a task which sets the level attribute's value and fires a simple event named levelchange at the BatteryManager object.

FIGURE 17.1 : Extrait des spécifications de l'API Battery Status.

Typiquement, une API est écrite sous forme de spécifications qui indiquent quelles propriétés et quelles méthodes sont disponibles pour les programmeurs, quels arguments peuvent être passés aux méthodes, et quelles sortes de valeurs sont retournées par les propriétés.

Dans le cas d'API développées par des institutions officielles, comme le consortium W3C (*World Wide Web Consortium*), il s'agit bien souvent de spécifications décrivant comment les programmes devraient être capables d'interagir avec les navigateurs Web, plutôt que la manière dont ils le font réellement. C'est alors aux équipes qui développent ces navigateurs Web de décider comment ils vont implémenter en pratique ces API.



Une API indique aux programmeurs comment ils peuvent interagir avec le logiciel, et comment celui-ci devrait répondre. Il est important de garder présent à l'esprit le fait que ce n'est pas parce qu'une API existe que les programmeurs peuvent l'utiliser. En fait, il existe de nombreuses API qui ont été proposées et décrites, mais qui n'ont pas encore été implémentées, ou qui ne l'ont été que partiellement, dans les navigateurs Web.

Vérifier le support par le navigateur des API HTML5

La meilleure source permettant de vérifier quel navigateur prend en charge un standard particulier, ou un standard simplement proposé, est le site `www.caniuse.com`. CanIUse.com liste tous les éléments et API HTML5, et il fournit un tableau des navigateurs qui les prennent en charge. La Figure 17.2 montre ce qu'il en est, à l'heure où ce livre est écrit, pour le standard IndexedDB.

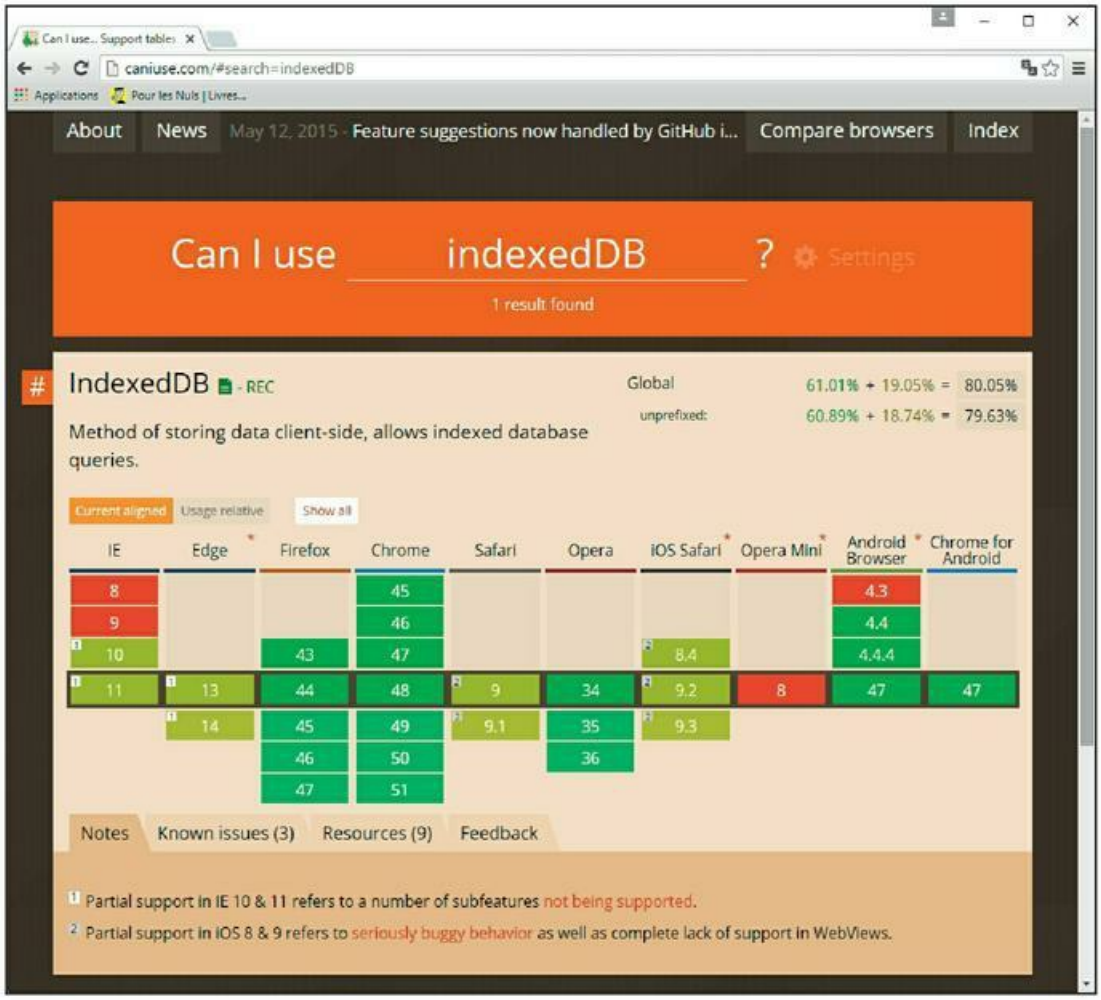


FIGURE 17.2 : L'état actuel du support de IndexedDB par les navigateurs Web, selon le site caniuse.com.

Découvrir les API de HTML5

Le standard HTML5 définit et documente nombre d'API que les programmeurs JavaScript peuvent utiliser pour accéder à des fonctionnalités des navigateurs Web d'une manière standardisée.

À ce jour, de nombreuses API HTML5 ne sont pas encore finalisées et implémentées par tous les navigateurs Web. Mais celles qui le sont se révèlent extrêmement utiles, et elles étendent les limites de ce qu'il est possible de faire avec des applications Web.

La liste des API de HTML5 change et s'étend constamment. Le Tableau 17.1 liste les plus populaires et les mieux prises en charge à ce jour.



Vous trouverez une liste un peu plus exhaustive des API de HTML5 sur le site compagnon de ce livre, à l'adresse `www.dummies.com/extras/codingwithjavascript`.

Tableau 17.1 : HTML5 et ses API.

API	Utilisation
Battery Status	Fournit des informations sur l'état de la batterie de l'appareil
Clipboard	Donne accès aux fonctions Copier, Couper et Coller du système d'exploitation
Drag and Drop	Prend en charge les opérations de glisser et déposer dans et entre les fenêtres de navigation

File	Fournit aux programmes un accès sécurisé au système de fichiers du dispositif
Forms	Fournit aux programmes un accès aux nouveaux types de données définis en HTML5
Geolocation	Donne accès aux applications Web aux données d'emplacement géographique du dispositif de l'utilisateur
getUserMedia/Stream	Fournit un accès aux données d'un dispositif externe, comme une webcam
Indexed database	Crée un système de base de données simple côté client dans le navigateur Web
Internationalization	Fournit un accès à des données de formatage localisées, ainsi qu'à des comparaisons de chaînes également localisées
Screen Orientation	Lit l'orientation de l'écran (portrait ou paysage), et donne aux programmeurs la possibilité de savoir quand cet état change afin de pouvoir réagir en conséquence
Selection	Prend en charge la sélection d'éléments en JavaScript en utilisant les sélecteurs de style CSS
Server-sent events	Permet au serveur de transmettre des données au navigateur sans que celui-ci ait besoin de les solliciter.
User Timing	Donne aux programmeurs accès à des outils temporels de haute précision afin de mesurer les performances des applications
Vibration	Donne accès au vibreur de l'appareil
Web Audio	Traite l'audio
Web Speech	Fournit des fonctionnalités pour la saisie vocale et la lecture du texte à voix haute
Web storage	Permet le stockage de couples clé/valeur dans le navigateur
Web sockets	Ouvre une session de communication interactive entre le navigateur et le serveur
Web workers	Permet à JavaScript d'exécuter des scripts en arrière-plan
XMLHTTPRequest 2	Améliore XMLHttpRequest pour éliminer le besoin de contourner les erreurs same-origin policy et lui permettre de travailler avec les nouvelles fonctionnalités de HTML5

Chaque API de HTML5 est conçue pour spécifier avec précision comment les programmeurs devraient être à même d'interagir avec les fonctionnalités des navigateurs Web ou des ordinateurs. Le processus de maturation menant de l'idée d'une API à la réalité de sa mise en œuvre peut cependant être un chemin très long et très complexe.

Plusieurs des API de HTML ont passé avec succès toutes les batteries de tests et de révisions leur permettant de devenir des standards bien pris en charge. Parmi celles-ci, la principale est l'API Geolocation (donc la géolocalisation).

Utiliser la géolocalisation

L'API Geolocation permet aux programmes d'accéder aux fonctions de géolocalisation du navigateur Web, et donc aux données indiquant quel est le lieu où se trouve l'appareil.

Cette API est certainement la mieux prise en charge de toutes, et elle est implémentée dans au moins 90 % des navigateurs mobiles ou de bureau, et donc dans tous les navigateurs les plus répandus, à l'exception d'Opera Mini.

Que fait la géolocalisation ?

L'API Geolocation décrit la manière dont JavaScript peut interagir avec l'objet `navigator.geolocation` de manière à obtenir des données sur la position courante d'un dispositif, notamment :

- **Latitude :** La latitude en degrés décimaux.
- **Longitude :** La longitude en degrés décimaux.
- **Altitude :** L'altitude en mètres.
- **Heading :** La direction dans laquelle pointe le dispositif.
- **Speed :** La vitesse à laquelle le dispositif se déplace en mètres par seconde.
- **Accuracy :** La précision avec laquelle la latitude et la longitude sont mesurées (en mètres).

En obtenant tout ou partie de ces données, une application JavaScript s'exécutant dans un navigateur Web pourra placer l'utilisateur sur une carte, faire appel à des ressources comme Google Maps afin de proposer à l'utilisateur des points d'intérêt ou des services proches de lui, et bien plus encore.

Comment la géolocalisation travaille-t-elle ?

Lorsque JavaScript initie une requête demandant la position de l'appareil *via* l'objet `Geolocation`, un certain nombre d'étapes sont exécutées avant que les informations de lieu ne soient renvoyées.

La première chose, si ce n'est la plus importante, est que le navigateur doit s'assurer que l'utilisateur a autorisé pour cette application Web spécifique l'accès à ses informations de localisation. Des navigateurs différents peuvent poser la question à l'utilisateur de différentes manières, mais il s'agit en général d'une sorte de notification ou de fenêtre pop-up. La Figure 17.3 illustre ce qui se passe dans Chrome lors d'un accès à un site marchand qui souhaiterait connaître votre position.

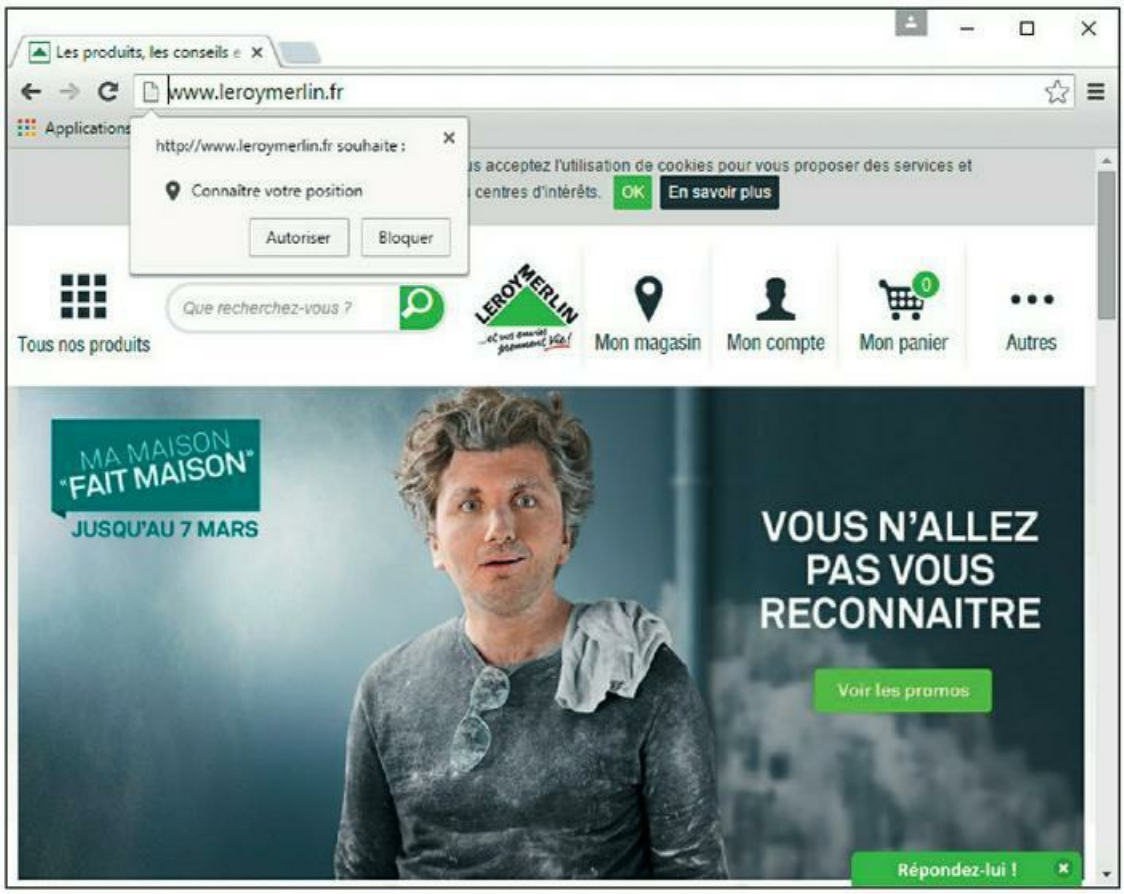


FIGURE 17.3 : Chrome affiche une requête de géolocalisation sous la barre d'adresse.

Si vous donnez votre accord, le navigateur va essayer de vous localiser. Il utilise pour cela différents moyens, en allant du plus précis au moins précis.

Si le programme indique qu'une précision élevée est nécessaire, la géolocalisation va prendre plus de temps pour essayer d'accéder aux données les plus précises disponibles *via* le GPS. Sinon, le navigateur va tenter de trouver le bon équilibre entre vitesse et précision afin d'obtenir les meilleurs résultats possibles en faisant appel aux ressources suivantes, si elles sont disponibles, bien sûr :

- la position des satellites GPS ;
- votre connexion réseau sans fil ;
- l'antenne à laquelle votre téléphone ou votre appareil mobile est connecté ;
- l'adresse IP de votre appareil ou de votre ordinateur.

Comment utiliser la géolocalisation ?

La clé pour utiliser la géolocalisation, c'est la méthode `getCurrentPosition` de l'objet `navigator.geolocation`. Cette méthode peut prendre trois arguments :

- `success` : Une fonction de rappel qui est passée à l'objet `Position` lorsque la localisation est un succès.
- `error` : Une fonction de rappel facultative qui est passée à l'objet `PositionError` lorsque la localisation échoue.

- `options` : Un objet PositionOptions facultatif qui peut être utilisé pour contrôler plusieurs aspects concernant la manière dont la recherche de la position est effectuée.

L'objet `Position` qui est retourné par la méthode `getCurrentPosition` contient deux propriétés :

- `Position.coords` : Contient un objet `Coordinates` qui décrit le lieu.
- `Position.timestamp` : L'instant auquel la position a été retrouvée.

Le Listing 17.1 vous montre comment vous pouvez utiliser la méthode `getCurrentPosition` pour obtenir l'objet `Position`, et parcourir les valeurs renvoyées dans `Position.coords`.

LISTING 17.1 : Obtenir des informations de localisation et les afficher dans le navigateur.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
    <title>L'objet Position</title>
      <script>
      var gps = navigator.geolocation.getCurrentPosition(

      function (position) {
      for (key in position.coords) {
          document.write(key+': '+ position.coords[key]);
        document.write ('<br>');
          }
        });
      </script>
  </head>
  <body>

  </body>
</html>
```

Si le dispositif sur lequel vous exécutez ce code prend en charge la localisation, et si le navigateur peut déterminer votre position, les résultats produits par ce script devraient à peu près ressembler à l'illustration de la Figure 17.4.

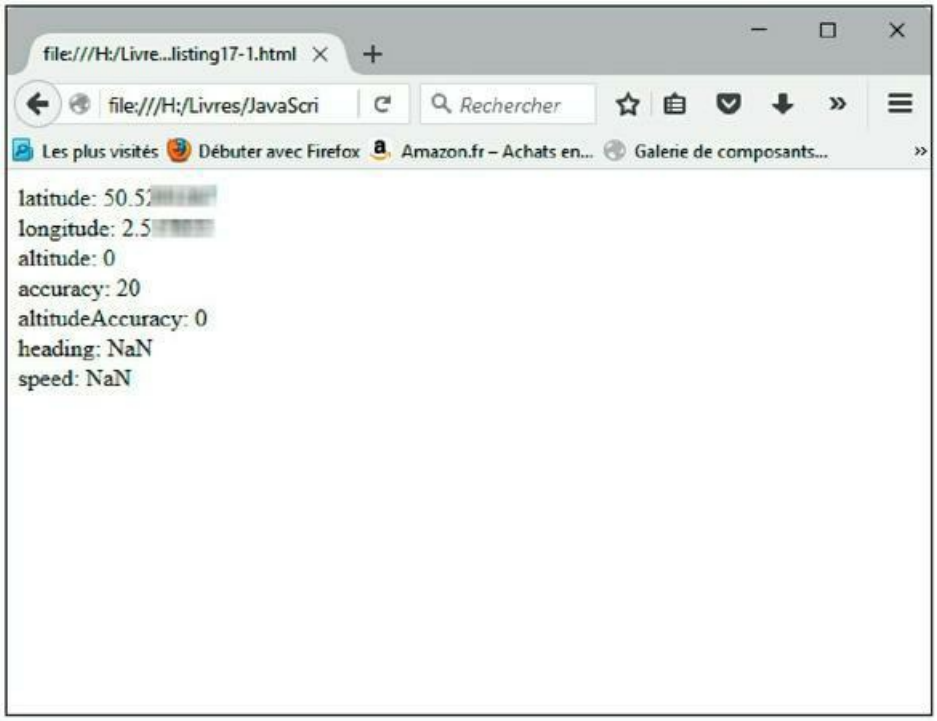


FIGURE 17.4 : Afficher les valeurs retournées dans l'objet Position.

Si le code est exécuté sur un ordinateur de bureau, certaines valeurs peuvent être nulles ou vides (0 ou `null`), voire indiquées comme NaN (*Not a Number*, donc pas un nombre) si elles apparaissent comme n'étant pas calculables. Bien entendu, l'exécution de ce script dans un navigateur mobile sur smartphone donnerait des résultats un peu différents, comme l'illustre la Figure 17.5.

Notez cette fois que les dernières valeurs obtenues sont `null`, cela venant du fait que l'appareil était stationnaire au moment où la capture écran a été réalisée.

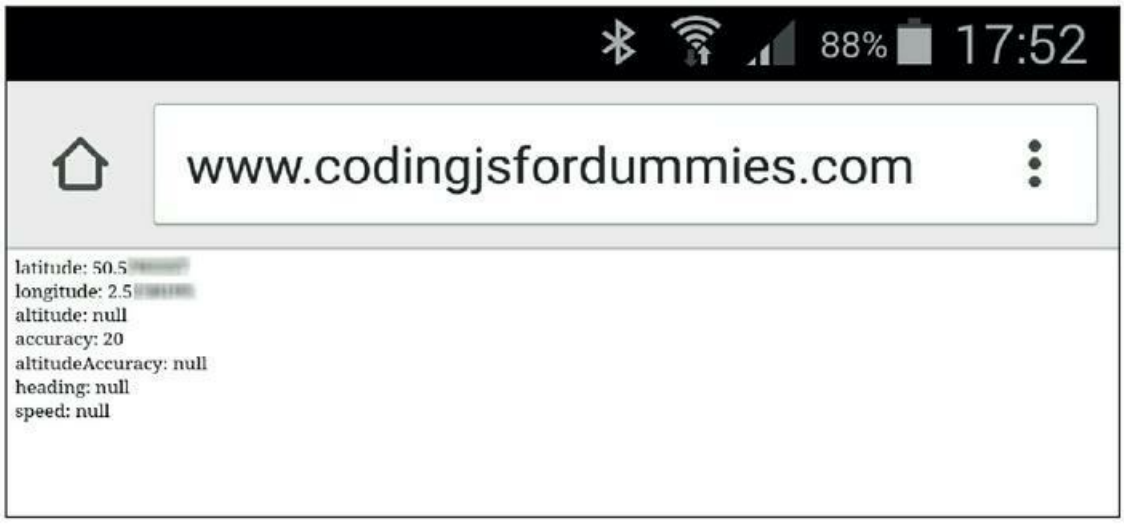


FIGURE 17.5 : Exécuter le script du Listing 17.1 sur un système mobile.

Combiner la géolocalisation avec Google Maps

L'une des applications les plus courantes de la géolocalisation consiste à afficher une position sur une carte. Pour y arriver, les premières choses dont vous avez besoin sont les informations de latitude et de longitude. Mais vous savez maintenant comment les obtenir. En revanche, dessiner une carte centrée sur une position donnée est un autre problème.

Fort heureusement, il y a des gens qui se sont déjà penchés sur cette question, et qui ont créé une API pour interagir avec leur logiciel de cartographie. Le fournisseur le plus célèbre, c'est évidemment Google, qui fournit gratuitement l'API permettant d'accéder à Google Maps (même si les motivations sous-jacentes sont commerciales).

Pour utiliser l'API de Google Maps, suivez ces étapes :

1. **Commencez par vous connecter à l'adresse `http://code.google.com`/ apis/console et connectez-vous avec votre compte Google.**
2. **Une fois connecté, il est possible que Google vous demande d'accepter son contrat de licence, ce que vous devez bien entendu faire.**
3. **Cliquez sur le lien qui indique Activer et gérer des API.**
4. **Donnez un nom à votre projet et cochez la case Oui pour accepter les conditions d'utilisation. Cliquez ensuite sur le bouton Créer.**

   Votre navigateur va afficher une liste des API disponibles ainsi qu'un champ de recherche.
5. **Dans la section API Google Maps, cliquez sur le lien qui indique Google Maps JavaScript API.**

   La page correspondante apparaît.
6. **Cliquez sur le bouton Enable pour activer l'API.**
7. **Cliquez ensuite sur le bouton Identifiants, dans le volet de gauche, puis sur Create credentials.**

   Une liste d'options vous est proposée.
8. **Cliquez sur l'option Clé API.**
9. **Dans la boîte de dialogue qui suit, cliquez sur le bouton Clé Navigateur.**
10. **Donnez un nom à la nouvelle clé puis cliquez sur le bouton Créer.**
11. **Au bout de quelques instants, vous allez obtenir votre clé API (voir la Figure 17.6).**

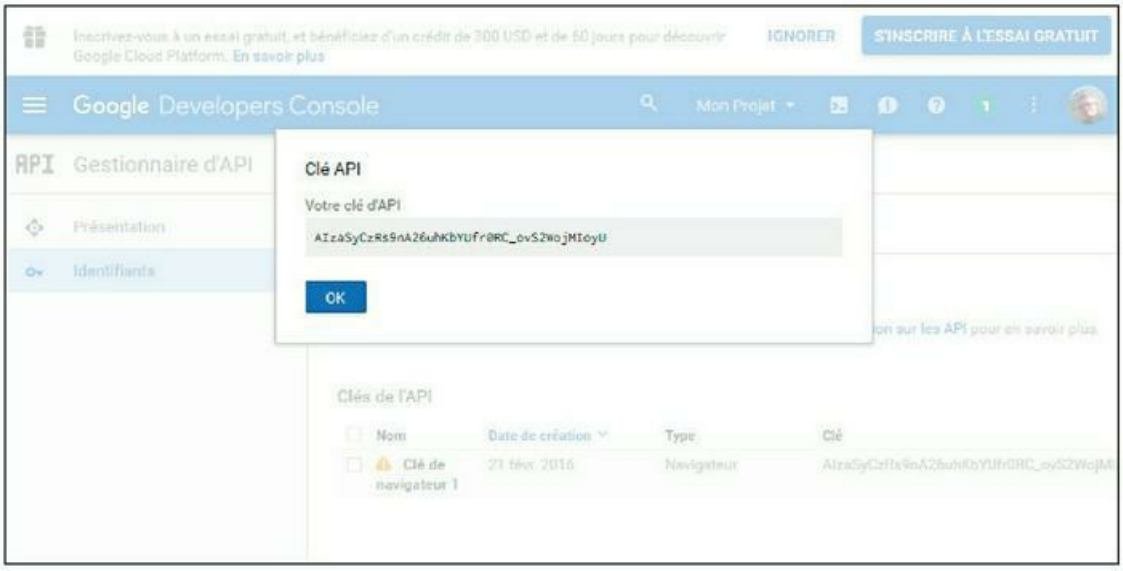


FIGURE 17.6 : Créer une clé pour l'API Google Maps JavaScript.

La page Identifiants affiche maintenant votre nouvelle clé API, qui est une longue série de lettres et de chiffres (voir la Figure 17.7).

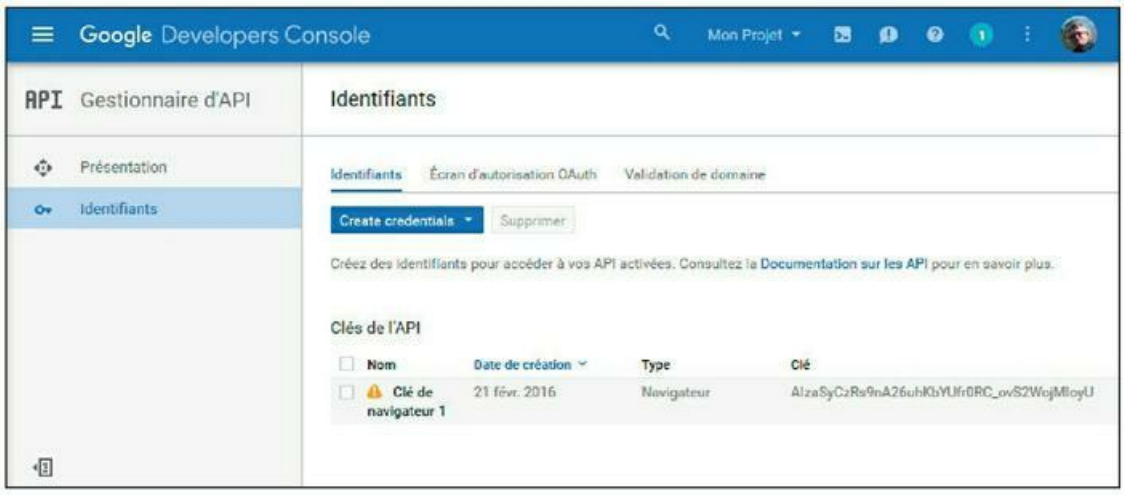


FIGURE 17.7 : Votre identifiant de clé API pour le navigateur est disponible.

Cette clé API est tout ce dont vous avez besoin pour accéder aux remarquables fonctionnalités de Google Maps. Il est temps maintenant de voir comment l'utiliser.

La page Web du Listing 17.2 récupère la position de votre ordinateur en utilisant l'objet `navigator.geolocation`, puis il la passe à Google Maps pour obtenir une carte. Remarquez sur le listing que l'emplacement du code où vous devez coller votre clé API est mis en surbrillance.

LISTING 17.2 : Associer la position obtenue avec l'API Geolocation et l'API de Google Maps.

```
<!DOCTYPE html>
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
    <title>Comment vous localiser</title>
      <style type="text/css">
```

```
        html, body, #map-canvas { height: 100%; margin: 0; padding: 0;}
      </style>

      <script type="text/javascript"
        src="https://maps.googleapis.com/maps/api/js?key=VOTRE_CLE_API">
      </script>

      <script>

      // exécute la fonction d'initialisation une fois maps chargé
      google.maps.event.addDomListener(window, ‘load', initialize);

      function initialize() {

      // récupère l'objet Position et l'envoie à une fonction de rappel
      var gps = navigator.geolocation.getCurrentPosition(

      // la fonction de rappel
      function (position) {
       //configure les options de Google Maps, en utilisant la latitude et
la
               longitude depuis l'objet Position
        var mapOptions = {
            center: { lat: position.coords.latitude, lng:
position.coords.
               longitude},zoom: 8
             };

          // construit la carte et la charge dans le div  map-canvas dans
la section
             <body>
          var map = new google.maps.Map(document.getElementById(‘map-
canvas'),
               mapOptions);
        }
         );
         };

      </script>
  </head>
  <body>
    <div id="map-canvas"></div>
  </body>
</html>
```

Pour que ce script fonctionne correctement, vous devez bien entendu remplacer le texte qui indique `VOTRE_CLE_API` par la clé API que vous avez obtenue de Google.

Le résultat de l'exécution du Listing 17.2 est illustré sur la Figure 17.8.

La copie d'écran de la Figure 17.8 a été réalisée avec Firefox, Chrome étant beaucoup plus intransigeant avec le traitement des fichiers locaux.

En modifiant légèrement le listing pour donner une valeur plus importante à l'option `zoom` (par exemple 16), vous obtiendrez une vue beaucoup plus rapprochée de votre position.

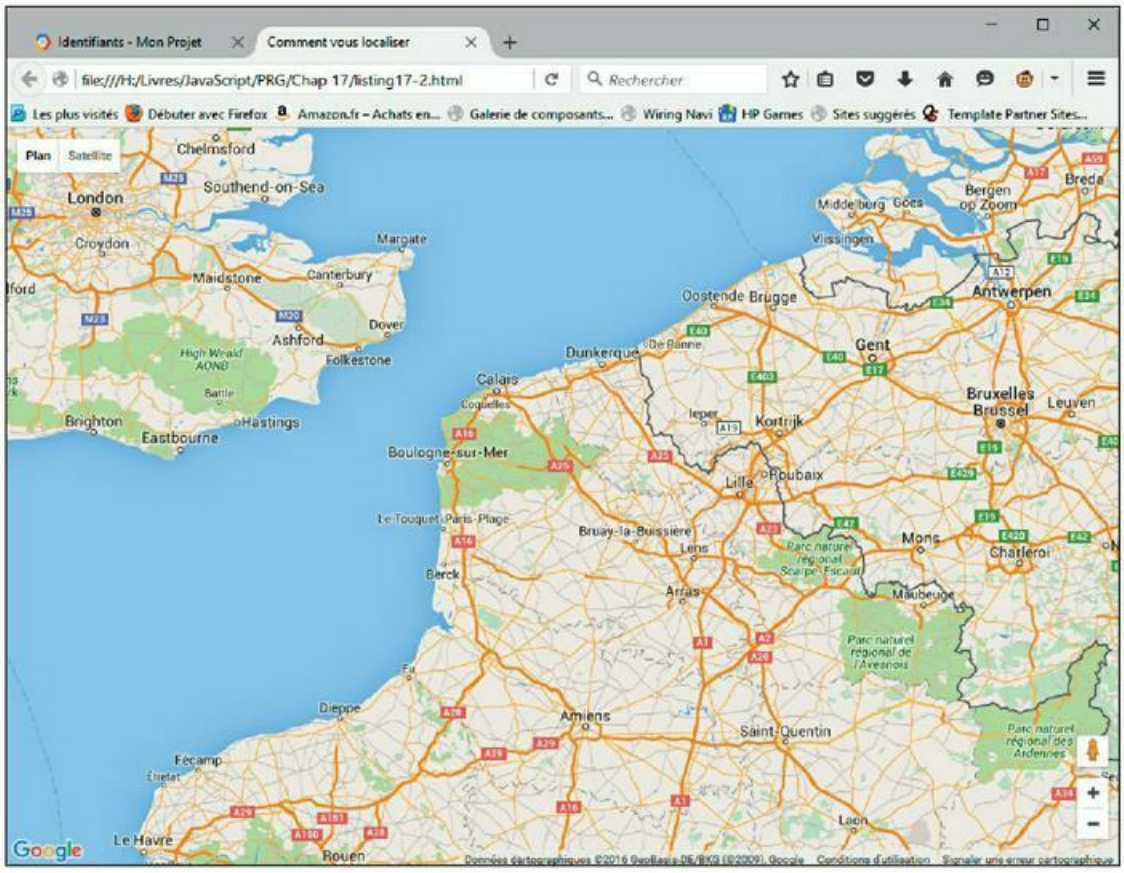


FIGURE 17.8 : Saurez-vous vous localiser ?

Accéder à l'audio et à la vidéo

Avant HTML5, la seule façon pour une page Web d'utiliser une caméra connectée à un ordinateur, ou intégrée à celui-ci, passait par l'utilisation de plug-ins spécialisés, notamment Flash.

L'un des objectifs majeurs poursuivis par HTML5 est d'éliminer le besoin de ces compléments, avec les mises à jour trop fréquentes et les problèmes de sécurité que cela pose. Depuis que HTML5 a été proposé, il y a eu plusieurs essais visant à définir un standard pour l'utilisation de ces caméras.

La dernière et la meilleure API destinée à gérer les communications audio et vidéo en direct *via* les navigateurs Web est appelée WebRTC (pour *Web Real Time Communications*, ou communications Web en temps réel).

Au cœur de WebRTC se trouve la méthode `navigator.getUserMedia ()`, qui permet d'obtenir un flux multimédia (audio et vidéo) à partir du dispositif dont se sert l'utilisateur.

Actuellement, `getUserMedia` est pris en charge par Chrome, Opera et Firefox. Si vous voulez l'utiliser dans d'autres navigateurs, comme Safari ou Internet Explorer, vous devrez utiliser un outil appelé un *polyfill*. Pour en



savoir plus sur les polyfills, voyez le site compagnon de ce livre, à l'adresse www.dummies.com/extras/codingwithjavascript.

Le premier paramètre de `getUserMedia` est un objet dont les propriétés indiquent le type de média auquel vous voulez accéder. Si vous avez besoin par exemple d'accéder aussi bien à la vidéo qu'à l'audio, vous devriez définir l'objet

suivant comme premier paramètre :

```
{video : true, audio : true}
```

Les autres paramètres demandés par `getUserMedia` sont une fonction de rappel en cas de succès, ou en cas d'échec. Le Listing 17.3 montre un exemple simple d'utilisation de getUserMedia.

LISTING 17.3 : Obtenir et afficher une ressource audio et vidéo de l'utilisateur.

```
<!DOCTYPE html>
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <head>
    <title>Lecture de média</title>
      <style type="text/css">
         html, body, #map-canvas { height: 100%; margin: 0; padding: 0;}
      </style>

  </head>
  <body>
    <video id = "v" autoplay></video>

    <script>
   window.addEventListener('DOMContentLoaded', function() {
   var v = document.getElementById('v');
   navigator.getUserMedia = (navigator.getUserMedia ||
                             navigator.webkitGetUserMedia ||
                             navigator.mozGetUserMedia ||
                              navigator.msGetUserMedia);
   if (navigator.getUserMedia) {
      // demande un accès à la vidéo seule
      navigator.getUserMedia(
         {
             video:true,
             audio:false
         },
         function(stream) {
            var url = window.URL || window.webkitURL;
          v.src = url ? url.createObjectURL(stream) : stream;
             v.play();
         },
         function(error) {
```

```
            alert('Il y a eu une erreur. (code erreur ' + error.code +
')');
             return;
         }
         );
   }
   else {
      alert('Désolé, votre navigateur ne supporte pas getUserMedia');
       return;
     };
});
    </script>
  </body>
</html>
```

Voyons de plus près ce que nous propose le Listing 17.3 :

```
window.addEventListener ('DOMContentLoaded', function () {
```

Il s'agit ici de l'événement principal. Il prend notamment en charge un gestionnaire d'événement qui attend que le DOM soit chargé avant d'exécuter la suite du script :

```
var v = document.getElementById ('v') ;
```

La ligne précédente crée une nouvelle variable, appelée `v`, qui contient une référence à l'élément vidéo possédant un identificateur appelé `id= "v"` :

```
navigator.getUserMedia = (navigator.getUserMedia ||
  navigator.webkitGetUserMedia ||
  navigator.mozGetUserMedia ||
  navigator.msGetUserMedia);
```

En fait, `getUserMedia` est une technologie expérimentale qui n'est pas complètement standardisée. De ce fait, les navigateurs Web peuvent l'implémenter de manière différente, ce qu'ils indiquent en appliquant leur propre préfixe. Cette instruction définit la valeur de l'objet `navigator.getUserMedia` selon la version préfixée prise en charge par le navigateur courant de l'utilisateur. Si, par exemple, vous utilisez Firefox et que vous appelez `getUserMedia`, vous allez appeler en fait `navigator.mozGetUserMedia` :

```
if (navigator.getUserMedia) {
```

Cela permet de vérifier si le navigateur de l'utilisateur prend en charge `getUserMedia` :

```
navigator.getUserMedia (
```

On appelle alors la méthode `getUserMedia` :

```
{
  video:true,
  audio:false
}
```

Le premier paramètre est un objet qui indique le type de média auquel vous voulez accéder :

```
function (stream) {
```

La fonction de rappel en cas de succès est exécutée si la requête `getUserMedia` est validée positivement. Elle prend un unique argument :

```
var url = window.URL || window.webkitURL;
v.src = url ? url.createObjectURL(stream) : stream;
```

Les deux lignes qui précèdent explicitent la manière dont deux navigateurs différents gèrent l'objet multimédia `stream`. La seconde ligne met en œuvre un opérateur ternaire en définissant la propriété `src` soit à la valeur `url. createObjectUrl (stream)`, soit à `stream`, selon la méthode prise en charge par le navigateur :

```
v.play () ;
```

Finalement, la vidéo est affichée. Si votre ordinateur prend en charge `getUserMedia`, et que vous avez une caméra (sinon, à quoi bon ?), vous pourrez voir une vidéo de vous-même (ou de ce vers quoi elle est dirigée) :

```
function(error) {
  alert('Il y a eu une erreur. (code erreur ' + error.code + ')');
  return;
}
```

Le code qui précède est envoyé par une fonction de rappel générique. Si le navigateur prend en charge `getUserMedia ()`, mais que l'utilisateur ne permet pas au navigateur d'accéder à la caméra, cette fonction va afficher un message d'erreur plus spécifique :

```
else {
    alert('Désolé, votre navigateur ne supporte pas getUserMedia');
  return;
};
```

La condition `else` affiche dans ce cas le message :

```
Désolé, votre navigateur ne supporte pas getUserMedia
```

Si le navigateur de l'utilisateur prend en charge getUserMedia, si celui-ci est équipé d'une caméra, et s'il autorise l'application à accéder à ce dispositif, une image vidéo en temps réel devrait s'afficher dans la fenêtre du navigateur, comme l'illustre la Figure 17.9.

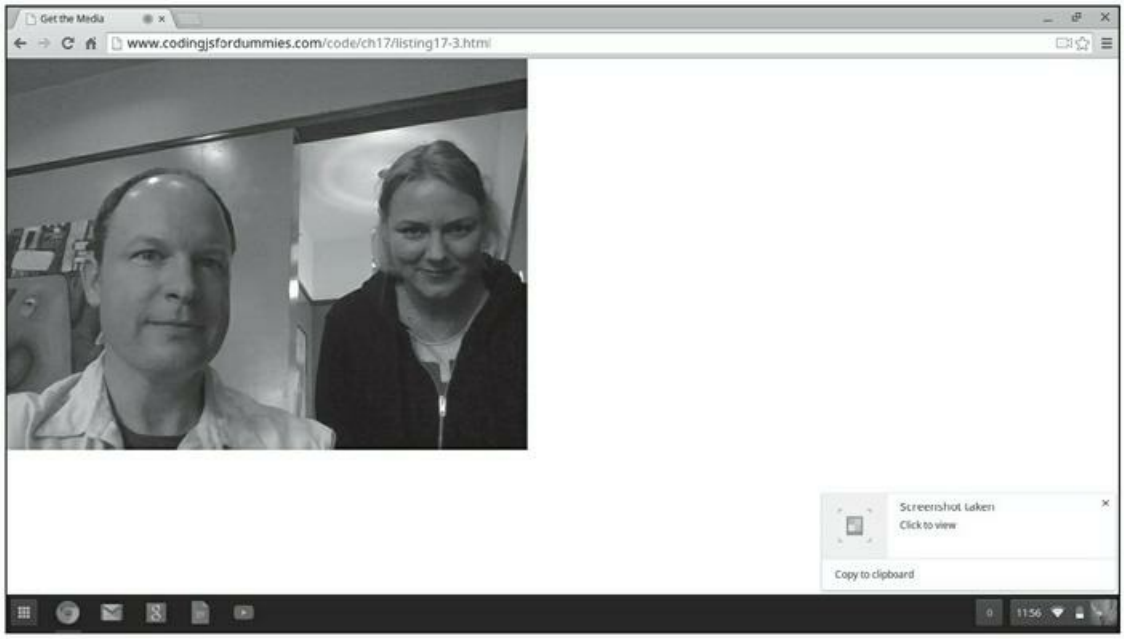


FIGURE 17.9 : Le navigateur permet d'afficher une vidéo en temps réel sans faire appel à un plug-in.

Chapitre 18
jQuery

DANS CE CHAPITRE :

- **Comprendre jQuery**
- **Sélectionner des éléments**
- **Créer des animations et des transitions avec jQuery**

« Il vaut mieux avoir vos outils avec vous. Sinon, vous aurez tendance à trouver quelque chose auquel vous ne vous attendiez pas, et vous serez découragé. »

Stephen King

jQuery est le compagnon le plus populaire de JavaScript, et il est utilisé par pratiquement tous les programmeurs pour accélérer et simplifier le développement de leurs applications JavaScript. Dans ce chapitre, vous allez découvrir les bases de jQuery et pourquoi il est si répandu.



N'oubliez pas de visiter le site Web compagnon de ce livre pour y télécharger les exemples de ce chapitre !

Écrire plus, tout en en faisant moins

jQuery est actuellement utilisé par plus de 61 % des 100 000 sites les plus visités. Il est si largement employé que de nombreuses personnes le voient comme un outil essentiel pour le développement en JavaScript.

jQuery permet de régler certains des problèmes les plus épineux en JavaScript, comme les questions liées à la compatibilité des navigateurs, et il rend la sélection ainsi que la modification de parties d'un document HTML bien plus faciles. jQuery contient également des outils vous permettant d'ajouter des animations et de l'interactivité à vos pages Web.

Les bases de jQuery sont faciles à apprendre une fois que vous connaissez JavaScript.

Débuter avec jQuery

Pour débuter avec jQuery, vous devez inclure sa bibliothèque dans vos pages Web. La méthode la plus simple consiste à utiliser une version hébergée sur un réseau de diffusion de contenu (ou RDC). Sinon, vous pouvez télécharger cette bibliothèque depuis le site de jQuery et l'héberger sur votre serveur.



Google héberge des versions de nombreuses bibliothèques JavaScript. Vous pouvez trouver des liens et copier des balises pour les coller dans votre code à l'adresse `http://developers.google.com/speed/library`.

Une fois que vous avez trouvé un lien pour une version disponible sur un réseau de diffusion de contenu (*Content Delivery Network*, ou CDN), il vous suffit de l'inclure entre les balises `<head>` et `</head>` de chaque page

dans laquelle vous avez besoin d'utiliser jQuery.



Il existe à l'heure actuelle deux branches de jQuery : la branche 1.x et la branche 2.x. La différence à retenir est que la dernière version de la branche 1.x est compatible avec Internet Explorer 6 à 8, tandis que la branche 2.x a éliminé le support de ces vieux navigateurs (de surcroît bogués).

Le Listing 18.1 illustre la manière d'incorporer jQuery dans une page Web.

LISTING 18.1 : Votre première page Web avec jQuery.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
<title>Hello JQuery</title>
<style>
  #helloDiv {
    background:  #333;
     color:  #fff;
    font-size:  24px;
    text-align: center;
    border-radius: 3px;
    width:  200px;
    height:  200px;
    display:  none;
  }
</style>
<script src="http://code.jquery.com/jquery-1.11.2.min.js"></script>
</head>
<body>

<button id="clickme">Cliquez-moi !</button>

<div id="helloDiv">Bonjour, JQuery !</div>

<script>
$( "#clickme" ).click(function () {
  if ( $( "#helloDiv" ).is( ":hidden" ) ) {
    $( "#helloDiv" ).slideDown( "slow" );
  } else {
    $( "div" ).hide();
  }
});
</script>
</body>
</html>
```

Une autre adresse possible pour lier votre code à jQuery est :



```
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js">
```

L'objet jQuery

Toutes les fonctionnalités de jQuery sont accessibles par l'intermédiaire de l'objet `jQuery`. Cet objet peut être référencé de deux manières différentes : par le mot-clé `jQuery`, ou par l'alias `$`. Les deux méthodes font absolument la même chose. Évidemment, l'emploi de l'alias est plus court, donc plus rapide, et c'est pourquoi il a la faveur des programmeurs.

La syntaxe de base pour utiliser jQuery se présente ainsi :

```
$ ("sélecteur").méthode () ;
```

La première partie, placée entre parenthèses, indique quels éléments vous voulez traiter. La seconde partie explicite ce qui devrait être fait avec ces éléments.

En réalité, les instructions jQuery exécutent souvent de multiples actions sur les éléments sélectionnés en utilisant une technique de *chaînage*. Celle-ci permet d'attacher plusieurs méthodes au sélecteur en ajoutant des points supplémentaires. Sur l'exemple du Listing 18.2, le chaînage est utilisé pour sélectionner un élément (ayant pour identificateur `pageHeader`) puis lui appliquer un style.

LISTING 18.2 : Utiliser le chaînage.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
  <title>Exemple de chaînage en jQuery</title>
  <script src="http://code.jquery.com/jquery-1.11.2.min.js"></script>
</head>
<body>
<div id="pageHeader"></div>
<script type="text/javascript">
        $("#pageHeader").text("Bonjour, le Monde !").css("color",
"red").css
("font-size", "60px");
</script>
</body>
</html>
```

Les méthodes chaînées de jQuery peuvent être assez longues, ce qui risque d'entraîner une certaine confusion. Cependant, vous ne devez pas oublier que JavaScript ne tient pas compte des espaces vides. Il est donc parfaitement possible de reformater l'instruction jQuery dans le Listing 18.2 pour la rendre plus lisible, comme ceci :

```
$("#pageHeader")
    .text("Bonjour, le Monde !")
```

```
    .css("color", "red")
    .css("font-size", "60px");
```

Est-ce que votre document est prêt ?

jQuery a sa propre manière d'indiquer que tout est bien chargé et opérationnel : l'événement *document ready*. Pour éviter des erreurs provoquées par le DOM, ou par le fait que jQuery pourrait ne pas être chargé au moment où le script est exécuté, il est important d'utiliser cet événement, à moins de placer toutes les instructions jQuery à la fin du document HTML (comme c'est le cas dans les Listings 18.1 et 18.2).

La syntaxe à utiliser est la suivante :

```
$(document).ready(fonction(){
  // les méthodes jQuery viennent ici. . .
});
```

Toutes les instructions jQuery que vous voulez faire exécuter une fois la page chargée doivent être placées dans cette structure. Les fonctions nommées peuvent bien entendu être définies ailleurs, puisqu'elles ne s'exécutent que quand elles sont appelées.

Utiliser les sélecteurs jQuery

À la différence des moyens compliqués, et limités, qu'offre JavaScript pour sélectionner des éléments, jQuery rend cette sélection simple. Avec jQuery, les programmeurs se servent des mêmes techniques que celles qu'ils utilisent pour sélectionner des éléments avec CSS. Le Tableau 18.1 liste les sélecteurs jQuery et CSS les plus courants.

Tableau 18.1 : Sélecteurs jQuery/CSS courants.

Sélecteur	Exemple HTML	Exemple jQuery
élément		$ ('p').css ('font-size', '12')
.class		$ ('.redtext').css
#id		$ ('#intro').fadeIn ('slow)
[attribut]		$ ('[data-role]').show ()

En plus de ces sélecteurs de base, vous pouvez modifier une section ou combiner des sélections de différentes manières. Par exemple, pour sélectionner le premier élément `p` d'un document, il est possible d'écrire :

```
$ ('p : first')
```

Pour sélectionner le dernier élément `p`, vous pouvez utiliser ceci :

```
$ ('p : last)
```

Pour sélectionner les éléments de liste numérotés qui sont impairs, vous pouvez utiliser :

```
$ ('li : odd')
```

Pour combiner des sélections multiples, il vous suffit de les séparer par une virgule. Par exemple, la ligne qui suit sélectionne tous les éléments `p`, `h1`, `h2` et `h3` d'un document :

```
$ (‘p, h1,h2,h3')
```

Il est donc possible de sélectionner des éléments de bien plus de façons qu’en pur JavaScript. Pour voir une liste complète de ces sélecteurs jQuery, vous pouvez vous reporter au site compagnon de ce livre www.dummies.com/extras/codingwithjavascript.

Changer des choses avec jQuery

Une fois votre sélection réalisée, l’étape suivante consiste à y modifier certaines choses. Les trois principales catégories de choses que vous pouvez modifier avec jQuery sont les attributs, les styles CSS et les éléments.

Lire et modifier des attributs

La méthode `attr ()` vous donne accès aux valeurs des attributs. Pour l’utiliser, il vous suffit de connaître le nom de l’attribut que vous voulez lire ou définir. Dans le code qui suit, la méthode `attr ()` est utilisée pour changer la valeur de l’attribut `href` d’un élément ayant pour identificateur “lien-accueil” :

```
$(‘a#lien-accueil').attr(‘href') =
    "http://www.codingjsfordummies.com/";
```

Cette instruction va changer le contenu de l’attribut `href` de l’élément sélectionné dans le modèle DOM. Lorsque l’utilisateur va cliquer sur le lien ainsi modifié, c’est la page Web spécifiée ici qui va s’ouvrir, et non le lien qui était associé à l’élément `img` dans le document source.



Modifier un élément en utilisant jQuery change uniquement sa représentation dans le DOM (et donc dans la fenêtre du navigateur de l’utilisateur). jQuery ne touche absolument pas au document HTML tel qu’il est enregistré sur le serveur. Si vous regardez le code source de la page Web, vous ne verrez d’ailleurs aucune modification.

Modifier des styles CSS

Changer les propriétés CSS avec jQuery ressemble de près à ce que nous avons vu dans le Chapitre 13 pour les propriétés de l’objet `Style`. jQuery rend ces modifications plus simples qu’avec du code JavaScript standard, et les propriétés de style s’écrivent exactement comme en CSS.

Le Listing 18.3 combine des changements de style à la volée avec des événements de formulaire pour permettre à l’utilisateur de contrôler la taille du texte.

LISTING 18.3 : Manipuler les styles avec jQuery.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
  <title>JQuery et CSS</title>
  <script src="http://code.jquery.com/jquery-1.11.2.min.js"></script>
  <script type="text/javascript">
    $(document).ready(function(){

      $('#sizer').change(function() {
        $('#theText').css('font-size',$('#sizer').val());
        });
     });
</script>
</head>
<body>
<div id="theText">Bonjour !</div>
<form id="controller">
  <input type="range" id="sizer" min="10" max="100">
</form>
</body>
</html>
```

L'exécution de ce code dans un navigateur est illustrée sur la Figure 18.1.



FIGURE 18.1 : Changer une propriété CSS avec un élément de saisie.

Manipuler des éléments du DOM

jQuery propose plusieurs méthodes pour changer le contenu des éléments, les déplacer, en ajouter, en supprimer et plus encore. Le Tableau 18.2 liste toutes les méthodes disponibles pour manipuler les éléments du DOM depuis jQuery.

Tableau 18.2 : Manipuler les éléments du DOM avec jQuery.

Méthode	Description	Exemple
text ()	Obtient le contenu des éléments trouvés, ou définit le contenu de ces éléments	$ ('p').text ('hello !')
html ()	Obtient la valeur du premier élément trouvé, ou définit le contenu de chaque élément trouvé	$ ('div').html (' Salut ')
val ()	Obtient la valeur du premier élément trouvé, ou définit le contenu de chaque élément trouvé	$ ('select#choices').val ()
append ()	Insère le contenu à la fin des éléments trouvés	$ ('div #closing').append (' Merci ')
prepend ()	Insère le contenu au début des éléments trouvés	$ ('dive #conclusion).prepend (' Pour faire ce que de droit. ')
before ()	Insère le contenu avant les éléments trouvés	$ ('#letter').before (header)
after ()	Insère le contenu après les éléments trouvés	$ ('#letter').after (footer)
remove ()	Supprime les éléments trouvés	$ ('.numTelephone').remove ()
empty ()	Supprime tous les nœuds enfants des éléments trouvés	$ ('.blackout').empty ()

Événements

Le Chapitre 11 discute des différentes méthodes permettant d'enregistrer des gestionnaires d'événements en JavaScript. Toutes restent parfaitement valides avec jQuery. Cependant, jQuery possède sa propre syntaxe pour gérer et enregistrer des événements.

La méthode proposée par jQuery, `on ()`, s'occupe des aspects complexes liés au fait de s'assurer que tous les navigateurs vont gérer les événements de la même manière, et elle demande également bien moins de saisie que les solutions purement JavaScript.

Utiliser on () pour attacher des événements

La méthode `on ()` de jQuery fonctionne pour l'essentiel de la même manière que `addEventListener ()`. Elle prend comme arguments un événement et une définition de fonction. Lorsque l'événement se déclenche sur le ou les éléments sélectionnés, la fonction est exécutée. Le Listing 18.4 utilise `on ()` ainsi qu'un sélecteur jQuery pour modifier la couleur de chaque ligne d'un tableau lorsque l'utilisateur clique sur le bouton.

LISTING 18.4 : Changer les couleurs d'un tableau en cliquant sur un bouton.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
```

```
  <title>jQuery CSS</title>
  <style>
  td {
    border: 1px solid black;
  }
  </style>
  <script src="http://code.jquery.com/jquery-1.11.2.min.js"></script>
  <script type="text/javascript">
    $(document).ready(function(){
      $(‘#colorizer').on(‘click',function() {
        $(‘#things tr:even').css(‘background','yellow');
        });

     });

</script>
</head>
<body>
<table id="things">
  <tr>
    <td>item 1</td>
    <td>item 2</td>
    <td>item 3</td>
  </tr>
  <tr>
    <td>pommes</td>
    <td>oranges</td>
    <td>citrons</td>
  </tr>
  <tr>
    <td>merlot</td>
    <td>malbec</td>
    <td>cabernet sauvignon</td>
  </tr>
</table>
<br>
<form id="tableControl">
  <button type="button" id="colorizer">Coloriser</button>
</form>
</body>
</html>
```

La Figure 18.2 illustre ce qui se passe une fois que l’utilisateur a cliqué sur le bouton.

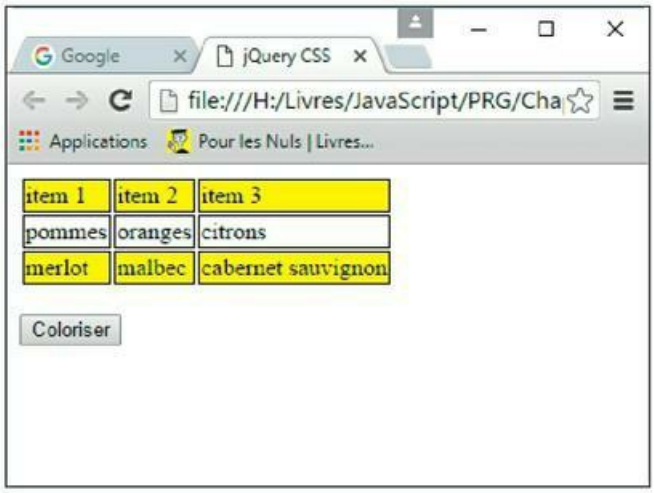


FIGURE 18.2 : Alterner les couleurs d'un tableau.



Vous avez peut-être remarqué quelque chose d'un peu étrange sur la colorisation des lignes du tableau. En effet, la première et la troisième ligne sont mises en couleur alors que le code jQuery demande à ce que ce soient les lignes paires (`even`) qui soient affichées en jaune. L'explication est simple : les déterminations pair ou impair sont basées sur le numéro d'indice de l'élément `tr`, numéro qui débute toujours à zéro. Les lignes colorisées sont donc la première (qui a pour rang 0) et la troisième (qui a pour rang 2).

Détacher des événements avec off ()

La méthode `off ()` peut être utilisée pour effacer ou désactiver un gestionnaire d'événement qui a été défini auparavant. Par exemple, si vous voulez désactiver le bouton du Listing 18.4 (peut-être parce que l'utilisateur devrait payer pour utiliser cette fonctionnalité), vous pourriez utiliser l'instruction suivante :

```
$ ('#colorizer').off ('click') ;
```

Si vous voulez désactiver tous les gestionnaires d'événement associés à un élément, il vous suffit d'appeler la méthode `off ()` sans argument :

```
$ ('colorizer').off () ;
```

Enregistrer des événements qui n'existent pas encore

Avec la nature dynamique du Web moderne, vous avez parfois besoin d'associer un gestionnaire d'événement à un élément qui est créé après que le code HTML a été chargé.

Pour réaliser cela, vous pouvez passer à la méthode `on ()` un sélecteur qui devrait être tracé de manière à détecter la présence de nouveaux éléments. Si, par exemple, vous voulez vous assurer que toutes les lignes actuelles, de même que toutes les futures lignes, d'un tableau sont cliquables, vous pourriez utiliser l'instruction suivante :

```
$(document).on('click','tr',function(){
  alert("Cliquable  !");
}
```

Autres méthodes pour les événements

En plus de `on ()`, jQuery possède également une syntaxe simplifiée pour attacher des gestionnaires d'événements aux éléments sélectionnés. Ces méthodes portent le même nom que les événements correspondants. Par exemple, ces deux instructions font la même chose :

```
$('#monBouton').on('click',function() {
  alert('Merci  !');
}
$('#monBouton').click(function() {
  alert('Merci  !');
```

Vous disposez notamment des raccourcis suivants :

- `change ()`
- `click ()`
- `dblclick ()`
- `focus ()`
- `hover ()`
- `keypress ()`
- `load ()`

Pour une liste complète des méthodes associées à des événements, reportez-vous au site de jQuery, à l'adresse http://api.jquery.com/category/events.

Effets

jQuery facilite grandement l'existence des programmeurs JavaScript. Et il en va de même pour la création d'effets et d'animations simples.



Les effets deviennent si simples avec jQuery qu'ils sont souvent surutilisés. Une fois que vous avez vu ce qui peut être fait, et que vous avez un peu de temps pour jouer avec les différentes variations possibles, une bonne idée consisterait à construire une application qui les utiliserait toutes chaque fois qu'un événement quelconque surviendrait. Quand vous en avez assez, supprimez ce fichier et dites-vous bien que cette surutilisation n'est pas une si bonne idée que cela, du moins du point de vue de l'utilisateur.

Effets de base

Les effets de base de jQuery contrôlent simplement si les éléments sélectionnés sont ou non affichés. Ces effets sont les suivants :

- `hide ()` : Masque les éléments sélectionnés.
- `show ()` : Montre les éléments sélectionnés.
- `toggle ()` : Bascule entre affichage et masquage des éléments sélectionnés. Un élément masqué sera réaffiché, et réciproquement.

Effets de transition

Vous pouvez ajouter un effet de transition aux éléments sélectionnés, ce qui permet de les rendre plus transparents ou plus opaques. Ces effets sont les suivants :

- `fadeIn ()` : L'élément sélectionné devient progressivement opaque selon la rapidité spécifiée pour l'effet.
- `fadeOut ()` : L'élément sélectionné devient progressivement transparent selon la rapidité spécifiée pour l'effet.
- `fadeTo ()` : Ajuste l'opacité de l'élément à une certaine valeur, et selon la rapidité spécifiée pour l'effet.
- `fadeToggle ()` : Rend les éléments sélectionnés plus opaques ou plus transparents selon la rapidité spécifiée pour l'effet.

Effets de glissement

Ces effets permettent de basculer progressivement entre visibilité et masquage, tout en les faisant glisser. Ces effets sont les suivants :

- `slideDown ()` : Affiche les éléments sélectionnés avec un effet de déplacement vers le bas.
- `slideUp ()` : Affiche les éléments sélectionnés avec un effet de déplacement vers le haut.
- `slideToggle ()` : Bascule le sens du déplacement.

Définir les arguments pour les méthodes d'animation

Chacune des méthodes d'animation de jQuery possède un jeu d'arguments qui contrôlent les détails de cette animation.

Les arguments des méthodes décrites ci-dessus sont les suivants :

- `duration` : Une valeur numérique indiquant la durée de l'animation (en millièmes de seconde).
- `easing` : Valeur chaîne indiquant un style à appliquer à l'animation. jQuery possède deux fonctions intégrées pour définir cet argument :
- `swing` : La progression de l'animation est un peu plus lente au début et à la fin qu'au milieu.
- `linear` : La vitesse de l'animation reste constante.
- `complete` : Spécifie une fonction à exécuter une fois l'animation courante terminée.

Appliquer des effets personnalisés avec animate ()

La méthode `animate ()` définit une animation personnalisée des propriétés CSS. Pour définir votre animation, vous passez un ensemble de propriétés à la méthode `animate ()`. Lorsqu'elle est exécutée, l'animation applique progressivement les valeurs de chaque propriété. Par exemple, pour modifier de cette manière la largeur et la couleur d'un élément `div`, vous pourriez utiliser cette instruction :

```
(‘div #monDiv').animate(
{
  width: 800,
  color: ‘blue'
}, 5000);
```

Ici, la largeur de l’élément `div` serait ajustée à 800 pixels, et sa couleur deviendrait bleue, la durée de l’animation étant de 5 secondes (5 000 millièmes de seconde).

Jouer avec les animations jQuery

Le Listing 18.5 implémente plusieurs des méthodes d’animation de jQuery. Vous pouvez bien entendu modifier les valeurs et faire des expériences avec les différents paramètres de chacune de ces méthodes pour voir ce qui se passe.

LISTING 18.5 : Jouer avec les animations jQuery.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
  <title>JQuery CSS</title>
  <style>
  td {
    border: 1px solid black;
  }
  </style>
  <script src="http://code.jquery.com/jquery-1.11.2.min.js"></script>
  <script type="text/javascript">
    $(document).ready(function(){
      $(‘#animator').on(‘click',function() {
        $(‘#items').fadeToggle(200);
        $(‘#fruits').slideUp(500);
        $(‘#wines').toggle(400,'swing',function(){
          $(‘#wines').toggle(400,'swing');
          });
        $(‘h1').hide();
        $(‘h1').slideDown(1000).animate({
          ‘color': ‘red',
          ‘font-size': ‘100px'},1000);
        });

     });

</script>
</head>
<body>
```

```
<h1>Voici un tas de choses !</h1>
<table id="things">
  <tr id="items">
    <td>item 1</td>
    <td>item 2</td>
    <td>item 3</td>
  </tr>
  <tr id="fruits">
    <td>pommes</td>
    <td>oranges</td>
    <td>citrons</td>
  </tr>
  <tr id="wines">
   <td>merlot</td>
    <td>malbec</td>
    <td>cabernet sauvignon</td>
  </tr>
</table>
<br>
<form id="tableControl">
  <button type="button" id="animator">Animez-moi !</button>
</form>
</body>
</html>
```

Ajax

L'une des choses les plus utiles, avec jQuery, est la manière dont il simplifie Ajax et rend le travail avec des données externes plus facile.

Ajax, qui est une technique permettant d'obtenir de nouvelles données dans une page Web sans avoir à recharger celle-ci, a été étudié dans le Chapitre 16. Ce chapitre a également traité de l'utilisation de données JSON dans JavaScript.

Utiliser la méthode ajax ()

Le point de départ des fonctionnalités Ajax de jQuery est la méthode ajax (). Cette méthode est la manière de bas niveau permettant d'envoyer et de retrouver des données depuis un fichier externe. Au niveau le plus simple, elle demande juste comme argument un nom de fichier ou une adresse URL et charge alors le fichier spécifié. Votre script peut ensuite affecter le contenu de ce fichier à une variable.

Vous pouvez également spécifier de nombreuses options différentes sur la façon dont l'URL externe devrait être appelée et chargée, ainsi que définir des fonctions qui devraient être exécutées si la requête réussit ou échoue.

Pour une liste complète des arguments facultatifs de la méthode `ajax ()`, visitez le site `http://api.jquery.com/jQuery.ajax`.

Sur le Listing 18.6, le script ouvre un fichier de texte contenant un paragraphe et affiche celui-ci dans un élément `div`.

LISTING 18.6 : Charger et afficher un fichier externe avec jQuery et AJAX.

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<head>
  <title>Introduction Dynamique</title>
  <script src="http://code.jquery.com/jquery-1.11.2.min.js"></script>

  <script>
  // attend que tout soit chargé
  $(document).ready(function(){
    // lorsque le bouton est cliqué
    $('#loadIt').on('click',function(){
      // récupère la valeur de la sélection et y ajoute .txt
      var fileToLoad = $('#intros').val() + '.txt';
      // ouvre ce fichier
      $.ajax({url:fileToLoad,success:function(result){
        // en cas de succès de l'ouverture, affiche le contenu du fichier
        $('#introtext').html(result);
       }});
     });
  });

  </script>
</head>
<body>
  <h1>Sélectionnez le type d'information voulu :</h1>
  <form id="intro-select">
    <select id="intros">
      <option value="none">Faites un choix</option>
      <option value="formal">Formel</option>
      <option value="friendly">Amical</option>
      <option value="piglatin">Charabia</option>
    </select>
    <button id="loadIt" type="button">Chargement !</button>
  </form>
  <div id="introtext"></div>
</body>
</html>
```



Si vous essayez d'exécuter le code du Listing 18.6 sur votre ordinateur local, les restrictions de sécurité appelées *same-origin policy* vont s'appliquer. Elles ne permettent pas de charger des données *via* Ajax, à moins qu'elles ne proviennent du même domaine (voyez à ce sujet le Chapitre 16). Pour essayer cet exemple, vous pouvez soit visiter notre site dédié (`http://www.codingjsfordummies.com/extras/coding with javascript`), télécharger les fichiers voulus sur votre serveur Web, ou encore désactiver les restrictions de sécurité du navigateur, comme cela est expliqué dans le Chapitre 16.

Raccourcis pour les méthodes Ajax

jQuery possède aussi plusieurs raccourcis pour les méthodes Ajax. La syntaxe de ces méthodes est simplifiée, car elles sont conçues pour des tâches spécifiques. Vous disposez des raccourcis suivants :

- `.get ()` : Charge des données sur un serveur en utilisant une requête HTTP GET.
- `.getJSON()` : Charge des données JSON sur un serveur en utilisant une requête HTTP GET.
- `.getScript ()` : Charge un fichier JavaScript sur un serveur en utilisant une requête HTTP GET et l'exécute.
- `.post ()` : Charge des données depuis un serveur et place le code HTML retourné dans l'élément correspondant.

Pour utiliser ces méthodes raccourcies, vous pouvez leur passer une URL, ainsi qu'un gestionnaire facultatif en cas de succès. Par exemple, pour obtenir un fichier sur un serveur en utilisant la méthode `.get ()`, puis insérer son contenu dans la page Web courante, vous pourriez écrire ceci :

```
$.get( "getdata.html", function( data ) {
  $( ".result" ).html( data );
});
```

Cet exemple est équivalent à l'instruction `.ajax ()` suivante :

```
$.ajax({
  url: getdata.html,
  success: function( data ) {
  $( ".result" ).html( data );
}
});
```

Certes, le gain n'est pas énorme ici. Mais, avec des requêtes Ajax plus complexes, apprendre et utiliser les raccourcis Ajax peut aider à produire plus rapidement un code plus compréhensible et plus concis.

Les Dix Commandements

DANS CETTE PARTIE...

Dix bibliothèques et frameworks pour aller plus loin avec JavaScript

Dix erreurs courantes avec JavaScript, et comment les éviter

Dix outils en ligne qui vous aideront à écrire du meilleur code JavaScript

Chapitre 19

Dix bibliothèques et frameworks pour aller plus loin avec JavaScript

DANS CE CHAPITRE :

- **Découvrir des bibliothèques et des frameworks JavaScript populaires**
- **Voir quels sites utilisent ces bibliothèques et ces frameworks**

« Je me tape la tête contre les murs, mais les murs reculent. »

Gustav Mahler

Vous venez tout juste de commencer votre voyage avec JavaScript. L'univers des outils, des *frameworks* et des bibliothèques dédiés à JavaScript et qui vous aident à écrire de meilleurs programmes est vaste, et il est en expansion permanente.

Dans ce chapitre, nous vous proposons de découvrir dix de nos *frameworks* et bibliothèques (librairies est également accepté) favoris. Vous n'avez pas besoin de tous et toutes les apprendre. Mais vous familiariser avec ces outils vous aidera énormément au cours de votre voyage avec JavaScript.



Il n'est pas forcément évident de définir la différence entre une bibliothèque et ce que l'on appelle un *framework*. Disons pour rester simple qu'une bibliothèque est un ensemble générique d'outils qui peuvent vous aider dans de multiples tâches de programmation. Un framework (on parle aussi de cadre d'applications) est aussi un ensemble d'outils, mais qui forment une certaine structure davantage dédiée à produire une architecture logicielle. Cela étant dit, il existe d'autres définitions de ces termes, et les débats ne sont pas clos…

Chacun de ses outils possède une base loyale d'utilisateurs, de fans et de programmeurs qui contribuent à son développement. À chaque fois, nous listerons quelques-uns de principaux sites qui les utilisent.

Angular JS

Angular JS, plus communément appelé Angular, est un framework d'application JavaScript open source (voir la Figure 19.1). Il est géré par Google et une communauté de développeurs.



FIGURE 19.1 : http://angularjs. org.

Le framework adapte et étend le HTML traditionnel pour fournir des contenus dynamiques *via* une liaison de données bidirectionnelle qui permet une synchronisation automatique entre modèles (données) et vues (pages Web). Angular JS permet en quelque sorte de masquer la manipulation du DOM avec comme objectif d'améliorer la testabilité et les performances du code.

En résumé, Angular JS a pour but :

- D'améliorer la testabilité du code en séparant manipulation du DOM et logique de l'application.
- De tester plus facilement le code.
- De créer une séparation entre le côté client de l'application et le côté serveur.
- De fournir une structure pour le processus de construction de l'application, de la conception initiale à l'interface utilisateur, en passant par la logique de l'écriture et le test du code.

Qui l'utilise ? YouTube.com, Lynda.com, Netflix.com ou encore freelancer.com.

Backbone.js

Backbone.js est une bibliothèque JavaScript MVC open source conçue pour créer des sites Web avec une page unique (voir la Figure 19.2). Le développement d'applications Web avec Backbone donne une structure à celles-ci, et renforce le très bon principe selon lequel la communication avec le serveur devrait être effectuée *via* une API RESTful.



MVC est un sigle qui signifie *Model-View-Controller*. Son principe consiste à séparer le modèle de données (*Model*) de l'interface utilisateur (*View*) en utilisant une couche de contrôle intermédiaire (*Controller*). Cette couche appelle le modèle et la vue voulus en fonction des données présentes en entrée. De son côté, REST signifie *Representational State Transfer* (transfert d'état représentationnel). Il s'agit en bref d'un style d'architecture pour le développement d'applications réseau s'appuyant sur un protocole de communications client-serveur sans état. Les applications RESTful utilisent typiquement des requêtes HTTP pour poster des données, les lire ou encore les supprimer.

Avec Backbone, votre code est plus modulaire. De plus, il vous permet de construire et de gérer des applications Web très complexes avec un minimum de code et une excellente organisation.

Backbone a une seule dépendance (`underscore.js`) et surcharge très peu le chargement de votre application Web.

Qui l'utilise ? reddit.com, bitbicket.org, tumblr.com, pintrest.com ou encore linkedin.com.


Backbone.js
backbonejs.org
Applications
Pour les Nuls | Livres...
Backbone.js (1.2.3)
» GitHub Repository
» Annotated Source
Getting Started
- Introduction
- Models and Views
- Collections
- API Integration
- Rendering
- Routing
Events
- on
- off
- trigger
- once
- listenTo
- stopListening
- listenToOnce
- Catalog of Built-in Events
Model
- extend
- constructor / initialize
- get
- set
- escape
- has
- unset
- clear
- id
- idAttribute
- cid
- attributes
- changed
- defaults
- toJSON
- sync
- fetch
- save
- destroy
- Underscore Methods (9)
- validate
- validationError
- isValid
- url
BACKBONE.JS
Backbone.js gives structure to web applications by providing models with key-value binding and custom events, collections with a rich API of enumerable functions, views with declarative event handling, and connects it all to your existing API over a RESTful JSON interface.
The project is hosted on GitHub, and the annotated source code is available, as well as an online test suite, an example application, a list of tutorials and a long list of real-world projects that use Backbone. Backbone is available for use under the MIT software license.
You can report bugs and discuss features on the GitHub issues page, on Freenode IRC in the #documentcloud channel, post questions to the Google Group, add pages to the wiki or send tweets to @documentcloud.
Backbone is an open-source component of DocumentCloud.
Downloads & Dependencies
(Right-click, and use "Save As")
Development Version (1.2.3)
69kb, Full source, tons of comments
Production Version (1.2.3)
7.3kb, Packed and gzipped
(Source Map)
Edge Version (master)
Unreleased, use at your own risk
build failing
Backbone's only hard dependency is Underscore.js (>= 1.7.0). For RESTful persistence and DOM manipulation with Backbone.View, include jQuery (>= 1.11.0), and json2.js for older Internet Explorer support. (Mimics of the Underscore and jQuery APIs, such

FIGURE 19.2 : http://backbonejs. org.

Ember.js

Ember.js est l'un des plus anciens frameworks JavaScript MVC, ses racines remontant à 2007. Il se définit lui-même comme « un framework pour créer des applications Web ambitieuses » (voir la Figure 19.3). Comme la plupart des autres frameworks décrits dans ce chapitre, il est basé sur une trame d'architecture logicielle MVC. Et, comme Backbone, il est conçu pour créer des applications Web monopages.

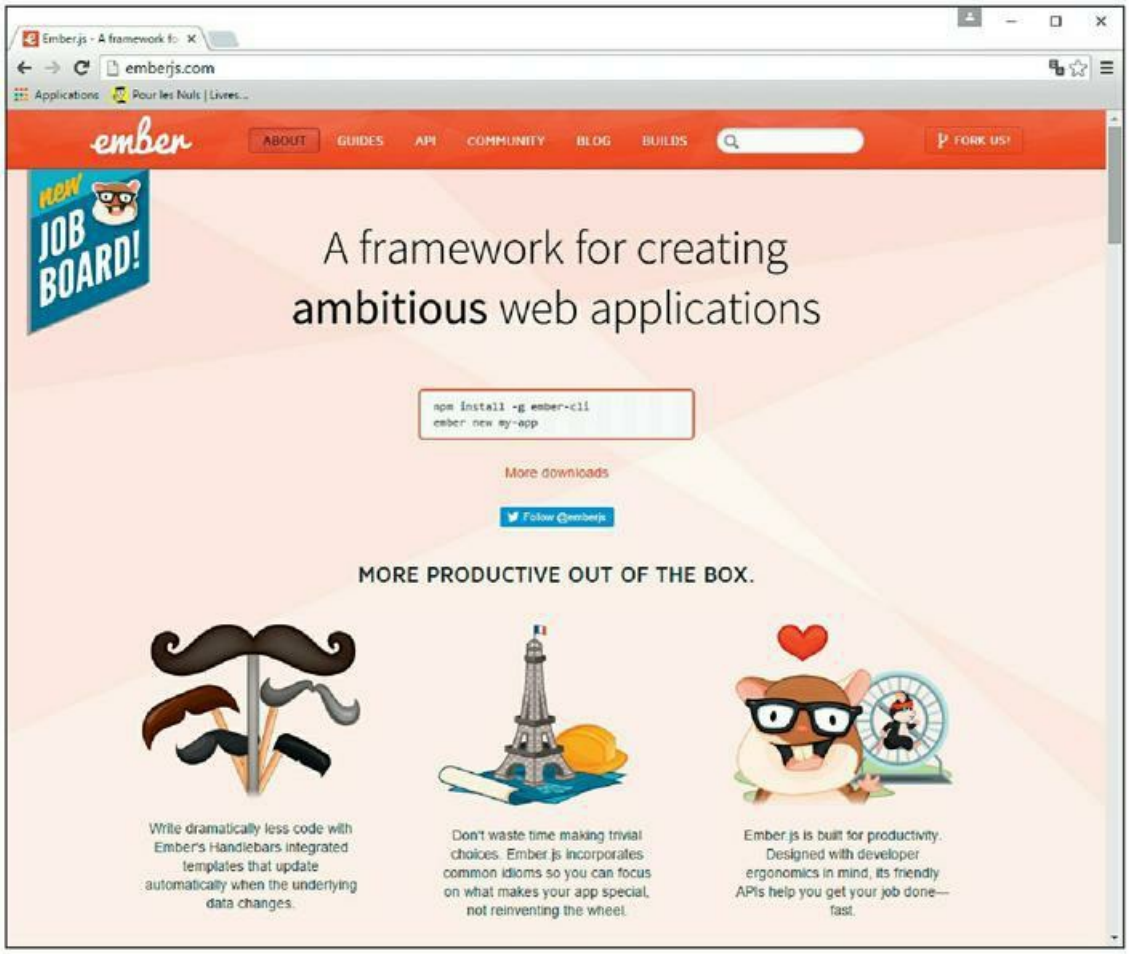

emberjs.com
ember
ABOUT
GUIDES
API
COMMUNITY
BLOG
BUILDS
FORK US!
new
JOB BOARD!
A framework for creating
ambitious web applications
npm install -g ember-cli
ember new my-app
More downloads
Follow @emberjs
MORE PRODUCTIVE OUT OF THE BOX.
Write dramatically less code with Ember's Handlebars integrated templates that update automatically when the underlying data changes.
Don't waste time making trivial choices. Ember.js incorporates common idioms so you can focus on what makes your app special, not reinventing the wheel.
Ember.js is built for productivity. Designed with developer ergonomics in mind, its friendly APIs help you get your job done—fast.

FIGURE 19.3 : http://emberjs.com.

Ember a la réputation d'être difficile à apprendre. Cependant, une fois que vous le connaissez, ses bénéfices sont multiples. Avec du code écrit selon les pratiques normales d'Ember, le développeur n'a plus besoin de tout spécifier manuellement pour son application, ce qui peut faire gagner beaucoup de temps.

Qui l'utilise ? digitalocean.com, vine.co, nbcnews.com, twitch.tv ou encore mediabistro.com.

Famo.us

Famo.us est un framework JavaScript open source servant à créer des interfaces utilisateur complexes pour n'importe quel écran (voir la Figure 19.4).

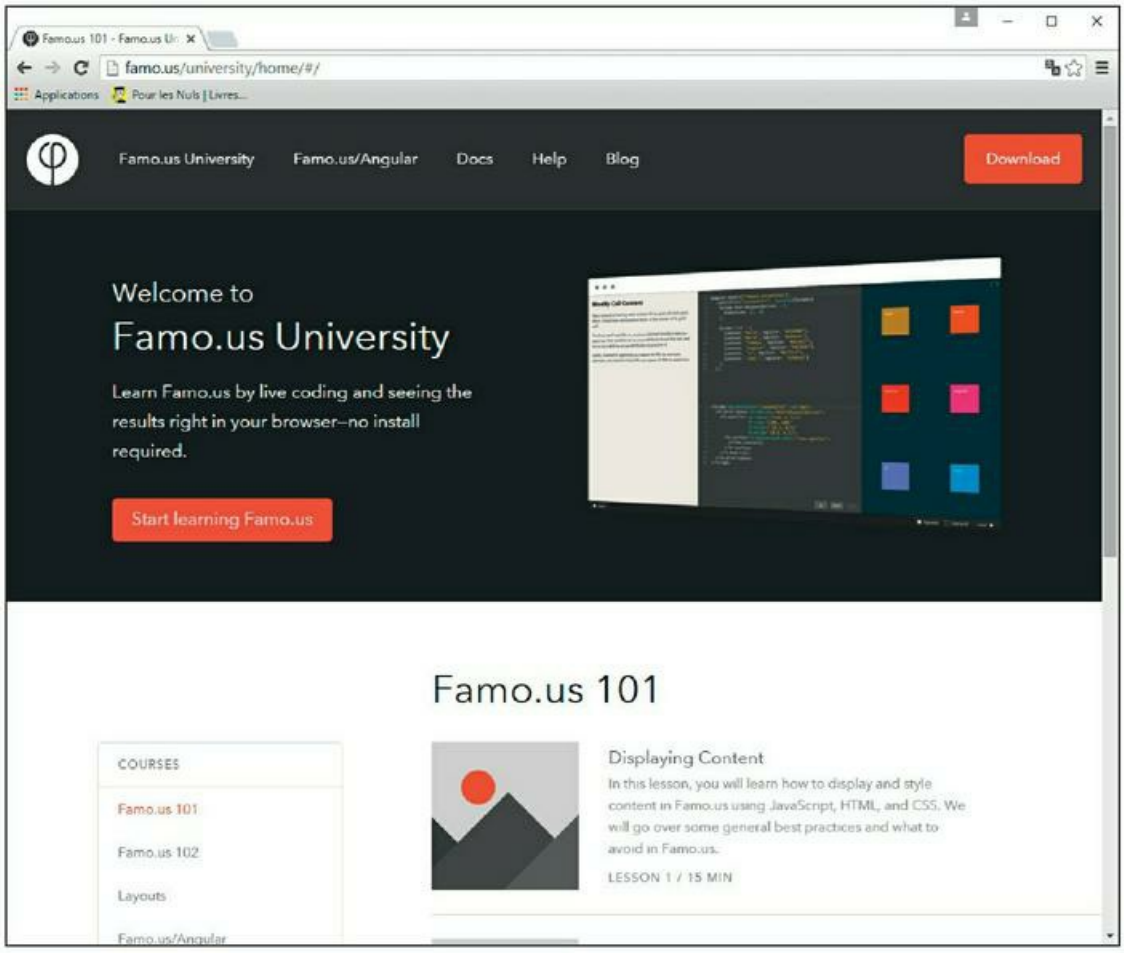

famo.us/university/home/#/
Famo.us University
Famo.us/Angular
Docs
Help
Blog
Download
Welcome to
Famo.us University
Learn Famo.us by live coding and seeing the results right in your browser–no install required.
Start learning Famo.us
Famo.us 101
COURSES
Famo.us 101
Famo.us 102
Layouts
Displaying Content
In this lesson, you will learn how to display and style content in Famo.us using JavaScript, HTML, and CSS. We will go over some general best practices and what to avoid in Famo.us.
LESSON 1 / 15 MIN

FIGURE 19.4 : http://famo.us.

Famo.us contient un moteur de rendu 3D, ce qui rend possible d'écrire du code JavaScript capable de déplacer des objets en 3D dans le navigateur, et de créer des effets et des interfaces qui n'étaient auparavant disponibles que dans des logiciels spécialisés. De ce fait, les applications créées avec Famo.us peuvent être plus rapides et plus fluides qu'avec une stricte utilisation de HTML5, CSS3 et JavaScript.

Qui l'utilise ? InkaBinka.com, SuperStereo, Requested App ou encore Japan Today.

Knockout

Knockout est un framework JavaScript open source servant à simplifier la programmation d'interfaces utilisateur dynamiques (voir la Figure 19.5). Il utilise un modèle dit MVVM (*Model-View-View-Model*) qui est un dérivé du modèle MVC.


Knockout : Home
knockoutjs.com
Applications
Pour les Nuls | Livres...
Home
Download / Install
Tutorials
Live examples
Documentation
Forum
Source
Knockout.
Simplify dynamic JavaScript UIs with the Model-View-View Model (MVVM) pattern
Download
v3.4.0 - 22kb min+gz
release notes
Key concepts
Declarative Bindings
Easily associate DOM elements with model data using a concise, readable syntax
Automatic UI Refresh
When your data model's state changes, your UI updates automatically
Dependency Tracking
Implicitly set up chains of relationships between model data, to transform and combine it
Templating
Quickly generate sophisticated, nested UIs as a function of your model data
More features
Free, open source (MIT license)
Pure JavaScript — works with any web framework
Small & lightweight — 54kb minified
... reduces to 20kb when using HTTP compression
No dependencies
Supports all mainstream browsers, even ancient ones
IE 6+, Firefox 3.5+, Chrome, Opera, Safari (desktop/mobile)
Fully documented
API docs, live examples, and interactive tutorials included
New: Interactive tutorials
Get started with knockout.js quickly, learning to build single-page applications, custom bindings and more with these interactive tutorials.
Live example
Run it:
Choose a ticket class:
Choose...
Clear

FIGURE 19.5 : http://knockoutjs.com.

Parmi les caractéristiques de Knockout, mentionnons celles-ci :

- liaisons déclaratives ;
- rafraîchissement automatique de l'interface utilisateur lorsque les données changent ;
- suivi des dépendances ;
- modélisation.

Qui l'utilise ? mlb.com, ancestry.com, Eventbrite.com ou encore ameritrade.com.

QUnit

QUnit est un framework destiné à la réalisation de tests pour JavaScript (voir la Figure 19.6). Il est utilisé pour de nombreux projets JavaScript open source, y compris jQuery. Il peut tester n'importe quel code JavaScript générique, et il est connu pour être aussi puissant que facile à utiliser.

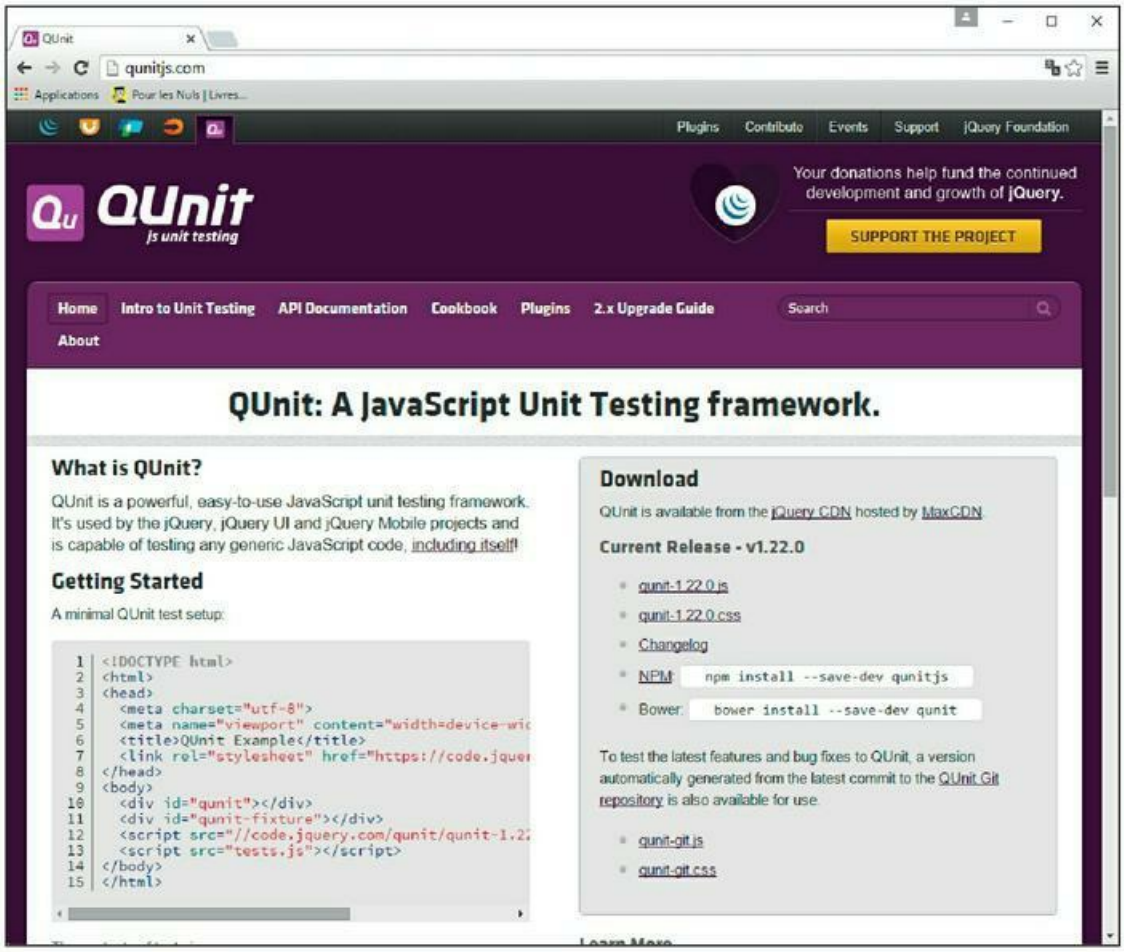


FIGURE 19.6 : http://qunitjs.com.

Qui l'utilise ? jQuery, jQuery UI, jQuery Mobile, sitepoint.com et de nombreux développeurs JavaScript.

Underscore.js

Underscore est une bibliothèque JavaScript qui fournit aux programmeurs de nombreuses fonctions utiles (voir la Figure 19.7). Une fois que vous aurez commencé à utiliser les fonctionnalités d'Underscore, vous vous demanderez comment vous aviez pu vivre sans elles.

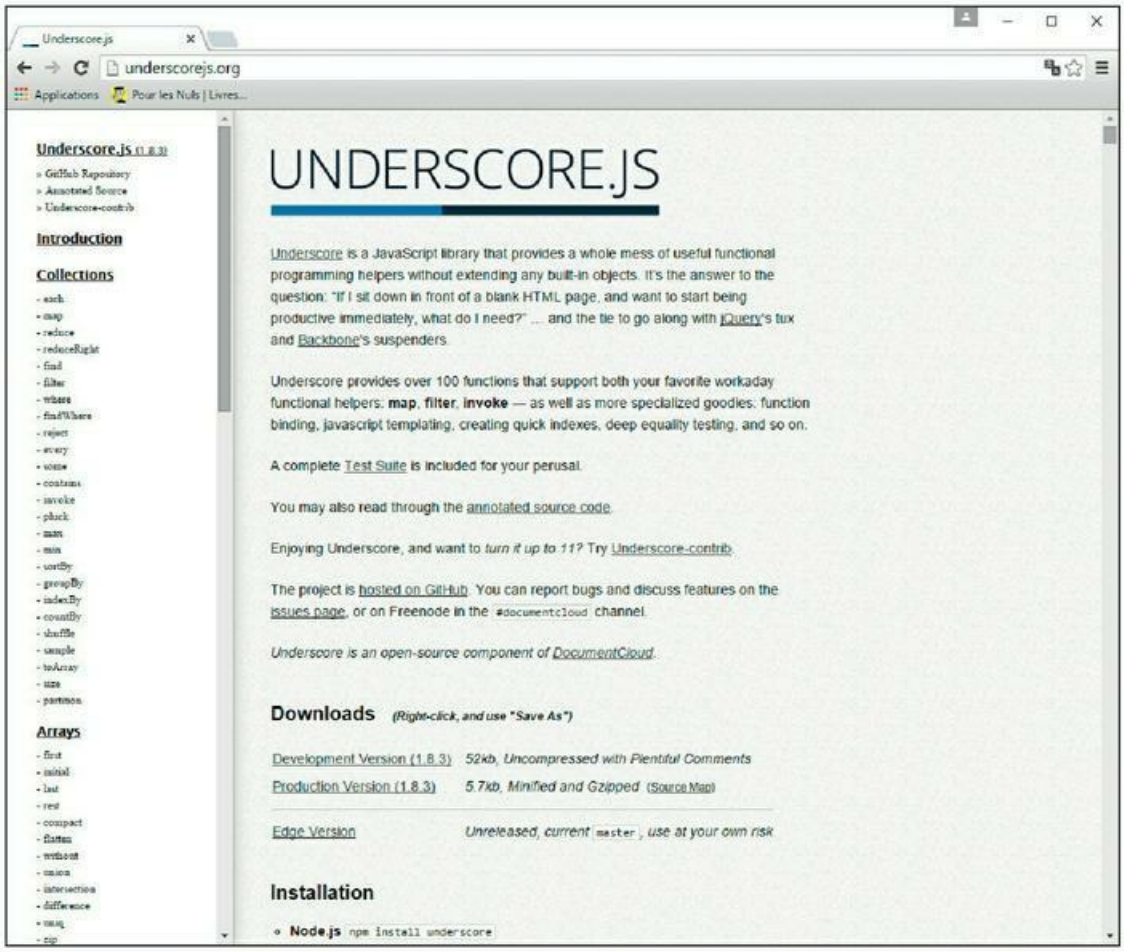

Underscore.js
underscorejs.org
Applications
Pour les Nuls | Livres...
Underscore.js (1.8.3)
» GitHub Repository
» Annotated Source
» Underscore-contrib
Introduction
Collections
- each
- map
- reduce
- reduceRight
- find
- filter
- where
- findWhere
- reject
- every
- some
- contains
- invoke
- pluck
- max
- min
- sortBy
- groupBy
- indexBy
- countBy
- shuffle
- sample
- toArray
- size
- partition
Arrays
- first
- initial
- last
- rest
- compact
- flatten
- without
- union
- intersection
- difference
- uniq
- zip
UNDERSCORE.JS
Underscore is a JavaScript library that provides a whole mess of useful functional programming helpers without extending any built-in objects. It's the answer to the question: "If I sit down in front of a blank HTML page, and want to start being productive immediately, what do I need?" ... and the tie to go along with jQuery's tux and Backbone's suspenders.
Underscore provides over 100 functions that support both your favorite workaday functional helpers: map, filter, invoke — as well as more specialized goodies: function binding, javascript templating, creating quick indexes, deep equality testing, and so on.
A complete Test Suite is included for your perusal.
You may also read through the annotated source code.
Enjoying Underscore, and want to turn it up to 11? Try Underscore-contrib.
The project is hosted on GitHub. You can report bugs and discuss features on the issues page, or on Freenode in the #documentcloud channel.
Underscore is an open-source component of DocumentCloud.
Downloads (Right-click, and use "Save As")
Development Version (1.8.3) 52kb, Uncompressed with Plentiful Comments
Production Version (1.8.3) 5.7kb, Minified and Gzipped (Source Map)
Edge Version Unreleased, current master, use at your own risk
Installation
Node.js npm install underscore

FIGURE 19.7 : http://underscorejs.com.

Parmi ces fonctionnalités offertes par Underscore, citons `sortBy` (pour trier des listes), `groupBy` (pour regrouper une collection en jeux d'éléments), `contains` (retourne `true` si une liste contient une valeur spécifiée), `shuffle` (retourne une copie mélangée d'une liste) et environ une centaine d'autres fonctions (dont la plupart auraient dû être implémentées dans JavaScript dès l'origine).

Qui l'utilise ? dropbox.com, lifehacker.com, theverge.com, att.com ou encore gawker.com.

Modernizr

Modernizr est une bibliothèque JavaScript permettant de détecter les fonctionnalités du navigateur Web dans lequel il est exécuté (voir la Figure 19.8).

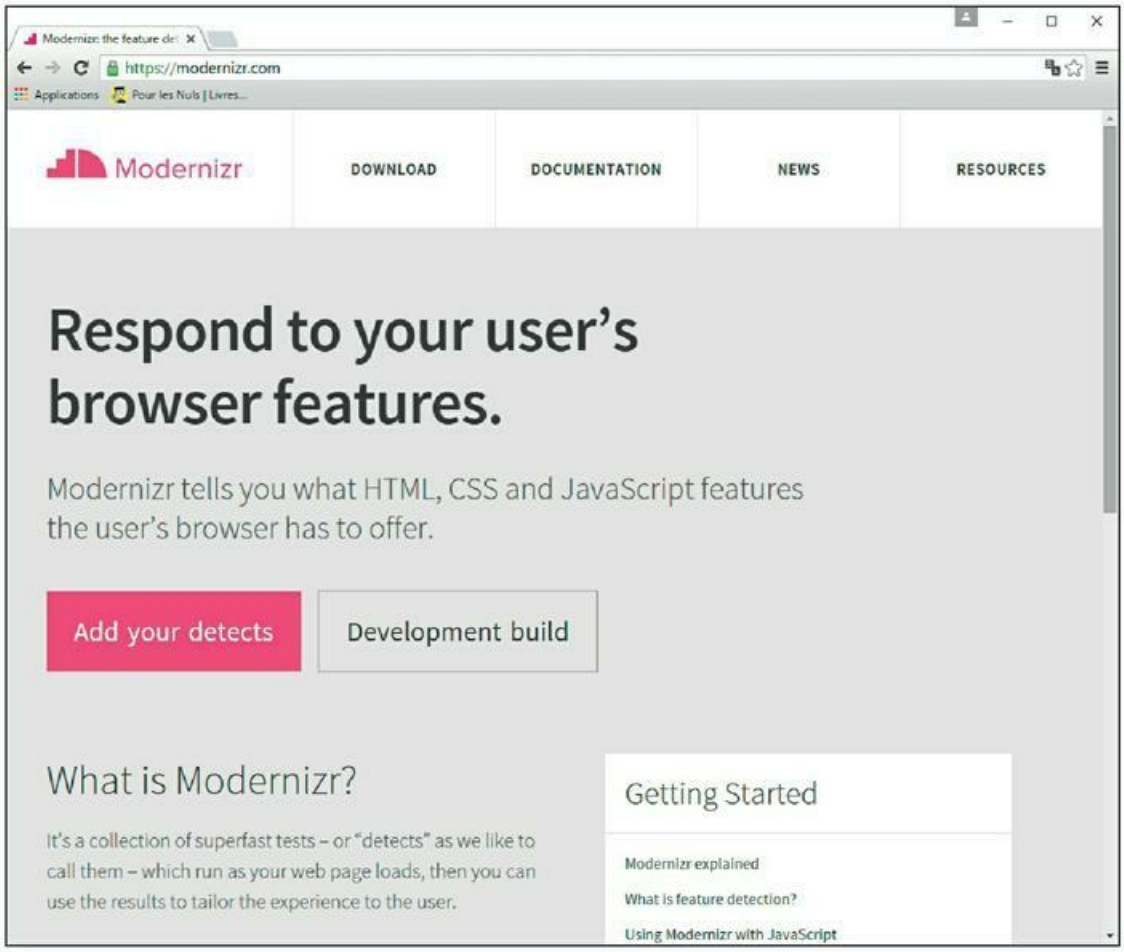

https://modernizr.com
Modernizr
DOWNLOAD
DOCUMENTATION
NEWS
RESOURCES
Respond to your user's browser features.
Modernizr tells you what HTML, CSS and JavaScript features the user's browser has to offer.
Add your detects
Development build
What is Modernizr?
It's a collection of superfast tests – or "detects" as we like to call them – which run as your web page loads, then you can use the results to tailor the experience to the user.
Getting Started
Modernizr explained
What is feature detection?
Using Modernizr with JavaScript

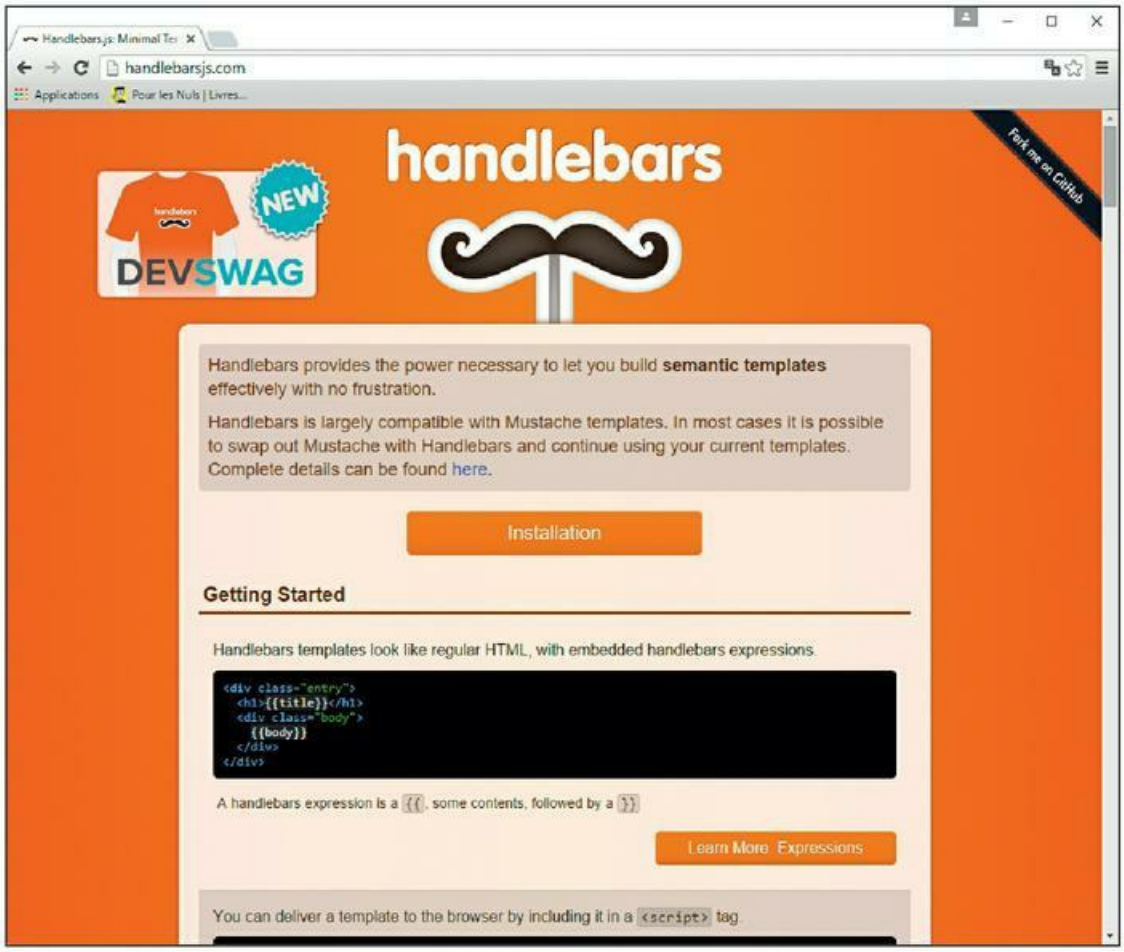
FIGURE 19.8 : http://modernizr. com.

Il est le plus souvent utilisé comme moyen très simple et pratique pour vérifier si le navigateur de l'utilisateur est capable d'exécuter un code JavaScript particulier, ou s'il faut utiliser d'abord une API avant d'essayer d'exécuter ce code. Modernizr est fréquemment employé en conjonction avec des outils appelés *Polyfills*, qui fournissent une méthode alternative pour mettre en œuvre certaines fonctionnalités avancées des navigateurs modernes sur des dispositifs ou dans des navigateurs moins évolués.

Qui l'utilise ? go.com, about.com, hostgator.com, addthis.com ou encore usatoday.com.

Handlebars.js

Handlebars est un moteur servant à créer des gabarits (ou *templates*) JavaScript côté client (voir la Figure 19.9). Il permet d'insérer ces gabarits dans des pages Web qui seront analysées afin d'utiliser les données en temps réel qui sont passées à Handlebars.js.


handlebarsjs.com
handlebars
NEW
DEVSWAG
Fork me on GitHub
Handlebars provides the power necessary to let you build semantic templates effectively with no frustration.
Handlebars is largely compatible with Mustache templates. In most cases it is possible to swap out Mustache with Handlebars and continue using your current templates. Complete details can be found here.
Installation
Getting Started
Handlebars templates look like regular HTML, with embedded handlebars expressions

<h1>{{title}}</h1>

{{body}}

A handlebars expression is a {{, some contents, followed by a }}
Learn More Expressions
You can deliver a template to the browser by including it in a <script> tag

FIGURE 19.9 : http://handlebarsjs.com.

Qui l'utilise ? meetup.com, mashable.com, flickr.com, wired.com ou encore overstock.com.

jQuery

jQuery est la bibliothèque « Écrire moins, en fait plus » de JavaScript (voir la Figure 19.10). jQuery est utilisé par plus de 60 % des sites les plus populaires du Web. Il est devenu un outil indispensable pour la plupart des programmeurs JavaScript. Quelques exemples des choses que jQuery facilite incluent la manipulation des documents, la gestion des événements, l'animation et Ajax.


jQuery
jquery.com
Applications
Pour les Nuls | Livres...
Plugins
Contribute
Events
Support
jQuery Foundation
jQuery
write less, do more.
Test with virtual machines on Mac, Linux or Windows, free.
Microsoft Edge
Try it
Download
API Documentation
Blog
Plugins
Browser Support
Search
Lightweight Footprint
Only 32kB minified and gzipped. Can also be included as an AMD module
CSS3 Compliant
Supports CSS3 selectors to find elements as well as in style property manipulation
Cross-Browser
IE, Firefox, Safari, Opera, Chrome, and more
Download jQuery
v1.12.1 or v2.2.1
View Source on GitHub →
How jQuery Works →
What is jQuery?
jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.
Corporate Members
WordPress
IBM
maxcdn
mt
Resources
jQuery Core API Documentation
jQuery Learning Center
jQuery Blog
Contribute to jQuery
About the jQuery Foundation
Browse or Submit jQuery Bugs
TRY jQuery
Learn jQuery in your browser!

FIGURE 19.10 : http://jquery.com.

De plus, jQuery a une architecture orientée plug-in, ce qui permet à d'autres développeurs d'étendre le noyau de ses fonctionnalités afin de créer de nouvelles bibliothèques et de nouveaux frameworks.

Parmi les plug-ins les plus connus de jQuery, mentionnons jQuery UI, jQuery Mobile, de nombreux effets, des sélecteurs de données, des outils de manipulation d'images, ou encore des outils dédiés aux diaporamas. Vous pourrez en trouver une liste complète en visitant le site `http://plugins.jquery.com`.

Qui l'utilise ? Wordpress.com, Pinterest, Amazon, Microsoft.com, Etsy et beaucoup, beaucoup d'autres.

Chapitre 20

Dix erreurs JavaScript courantes et comment les éviter

DANS CE CHAPITRE :

- **Éviter les crochets déséquilibrés**
- **Se méfier de la ponctuation incorrecte**
- **Réparer les erreurs**
- **Ajuster les mauvais noms de variables**

« N'ayez pas peur de la perfection – vous ne l'atteindrez jamais. »

Salvador Dali

Même les meilleurs programmeurs JavaScript font des erreurs. Parfois, ces erreurs font que votre programme ne fournit pas les résultats attendus, et parfois aussi qu'il ne fonctionne pas du tout. Dans tous les cas, il y a (au moins) une erreur, c'est-à-dire un *bogue* (la version francisée de *bug*, ou insecte). Tout au long de ce livre, nous vous avez fourni des astuces et des outils pour trouver et corriger vos erreurs.

Pour devenir un meilleur programmeur, vous devez être capable d'identifier les sources potentielles d'erreurs pour les réparer plus facilement et plus rapidement. Avec un certain entraînement, vous remarquerez que vous faites de moins en moins d'erreurs, et que ce joli code sans bogues coule de plus en plus facilement de vos doigts agiles. Au fil du temps, vous allez commencer à devenir un vrai Ninja du JavaScript.

Dans ce chapitre, nous allons passer en revue dix erreurs que les programmeurs de tous niveaux commettent souvent. Nous vous donnerons évidemment des conseils pour les éviter.

Confusion dans l'égalité

Est-ce que *x* est égal à *y* ? Est-ce que *x* est vrai (`true`) ? Les questions d'égalité sont centrales en JavaScript et peuvent sembler assez confuses. Tout cela tourne autour de trois zones de JavaScript : les instructions et opérateurs conditionnels (`if`, `& &` et ainsi de suite), l'opérateur d'égalité (==) et l'opérateur d'égalité stricte (===).

Pour nous compliquer encore plus l'existence, l'opérateur d'affectation (=) ressemble furieusement à ce que la plupart d'entre nous appellent le signe égal. Ne vous trompez pas ! Voici donc un rapide tour d'horizon avec des exemples, les situations où chacun de ces opérateurs (=, == et ===) est approprié et utile.

Éviter une mauvaise utilisation de l'affectation

L'opérateur d'affectation assigne l'opérande de droite à l'opérande de gauche. Par exemple :

```
var a = 3 ;
```

Ici, l'instruction affecte à une nouvelle variable, appelée `a`, la valeur 3.

Dans un programme, un *opérande* peut représenter n'importe quoi. C'est un peu comme un sujet ou un substantif dans une phrase, les opérateurs (+, -, *, / et ainsi de suite) représentant les verbes.

Les opérateurs d'affectation peuvent également comporter des expressions (parfois complexes) sur leur côté droit. Ces expressions sont évaluées en premier, puis assignées à l'opérande de gauche.

Une erreur courante des débutants consiste à écrire une affectation dans une instruction de comparaison. Par exemple :

```
if (a = 4) {… }
```

Si vous pensiez comparer la valeur de la variable a avec le chiffre 4, vous pouvez être certain que ce code ne donnera pas le résultat escompté.

L'égalité et ses écueils

L'opérateur d'égalité (==) et son double diabolique, l'opérateur d'inégalité (! =), peut être plutôt souple, mais aussi très dangereux. Nous vous conseillons de l'utiliser le moins possible. Voici pourquoi :

```
0 == '0'
```

Toute personne ayant passé un peu de temps à programmer sait qu'un nombre placé entre des apostrophes n'est en réalité pas un nombre. Mais l'opérateur `==` considère qu'il s'agit de la même chose, car il effectue une conversion de type avant de comparer les valeurs. Cela peut conduire à toutes sortes de problèmes difficiles à détecter.

Si vous voulez comparer une chaîne avec un nombre et obtenir comme réponse `true` si les deux semblent identiques, il est plus sûr de le faire de manière explicite comme ceci :

```
parseInt (0) === parseInt ("0")
```

Cette instruction est également évaluée comme étant `true`, mais sans invoquer un genre de magie vaudou. Cela nous amène à nos amis, l'égalité stricte (===) et l'inégalité stricte (! ==). Eux font exactement ce que vous attendez. Quel est à votre avis le résultat renvoyé par l'instruction suivante :

```
0 === '0'
```

Vous avez dit `false` ? Vous avez parfaitement raison. Les deux opérandes sont en effet clairement de type différent.

Les crochets mal accrochés

Plus un programme devient compliqué, et c'est en particulier le cas lorsque vous travaillez avec des objets JavaScript, et plus les crochets commencent à s'empiler. C'est là que vous commencez à voir des comportements étranges, ou des erreurs ésotériques, dans votre console JavaScript.

Voici un exemple de code JavaScript où les crochets ne sont pas correctement appairés :

```
{
    "status": "OK",
    "results": [{
    "id":  12,
    "title": "Coder en JavaScript pour les Nuls",
    "author": "Chris Minnick and Eva Holland",
    "publication_date": "",
   "summary_short": "",
    "link":  {
```

```
        "type": "review",
         "url": "",
        "link_text": "Lisez les avis sur Coder en JavaScript pour les
Nuls"
    },
    "awards": [{
        "type": "Prix Nobel",
         "url": "",
    }]
}
```

Est-ce que vous arrivez à localiser le problème ? Compter le nombre de crochets ou d'accolades pour trouver celui ou celle qui est en trop ou manquant, peut prendre pas mal de temps. Et si vous n'y arrivez pas, vous avez un sérieux souci !

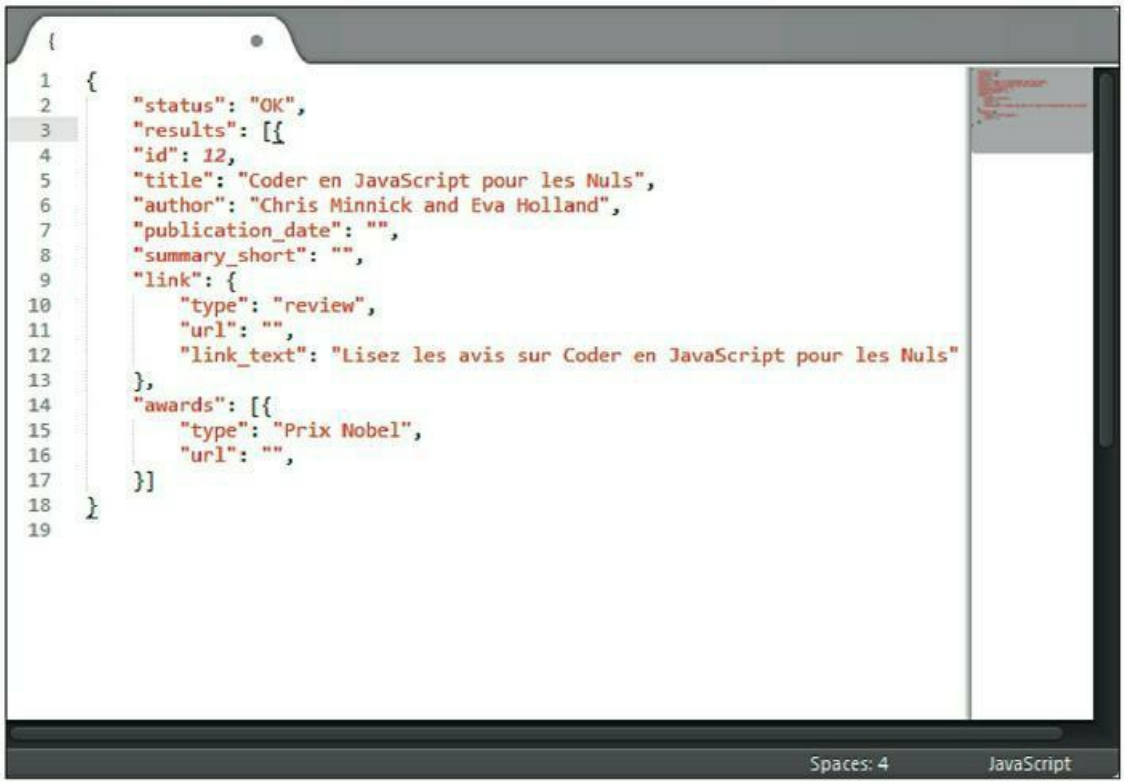


FIGURE 20.1 : Subliment Text met en évidence les crochets (ou les accolades) ouvrant et fermant.

Dans ce cas, un bon éditeur de texte peut être extrêmement précieux. Par exemple, Sublime Text a une fonctionnalité qui vous montre la correspondance entre les crochets et les accolades (ou du moins ce qu'il pense être cette correspondance) en soulignant le couple trouvé lorsque vous placez le pointeur juste après un crochet (ou une accolade) ouvrant ou fermant. C'est ce qu'illustre la Figure 20.1.

Confusion entre guillemets et apostrophes

JavaScript vous permet d'utiliser au choix des guillemets ou des apostrophes pour définir les chaînes de caractères. Par contre, il est totalement dénué de souplesse pour ce qui concerne le respect de la règle qui dit que guillemet va avec guillemet, et apostrophe avec apostrophe. C'est là aussi une erreur courante, et donc l'une des premières choses à regarder en cas de problème. Par exemple :

```
var movieName = "Popeye'; // erreur !
```

```
var welcomeMessage = ‘Merci, ‘ + firstName + ‘, d'apprendre JavaScript !'
//
erreur  !
```

Les parenthèses manquantes

Cette erreur se rencontre le plus souvent dans les instructions conditionnelles, en particulier celles qui comportent de multiples conditions. Par exemple :

```
if (x > y) && (y < 1000) {
...
}
```

Ce que nous voulons faire ici, c’est vérifier si les deux conditions sont vraies. Cependant, il y a ici trois conditions au travail, et toutes nécessitent des parenthèses. Ce qui manque dans l’exemple précédent, ce sont donc les parenthèses autour de la condition ultime, `& &`, celle qui dit que les deux autres doivent être vérifiées pour que le code traite les instructions placées entre les accolades.

Pour que ce code soit correct, il faut donc l’écrire de la manière suivante :

```
if ((x > y) && (y < 1000)) {
...
}
```

Le point-virgule manquant

Les instructions JavaScript devraient toujours se terminer par un point-virgule. Cependant, si vous placez chaque instruction sur sa propre ligne et que vous n’ajoutez pas de point-virgule, JavaScript traitera tout de même ce code comme si ces signes de ponctuation étaient là. Même si l’exécution semble se dérouler normalement, l’absence de point-virgule peut conduire à de sérieux problèmes si vous voulez réorganiser votre code, ou si deux instructions finissent par se terminer sur la même ligne.

La meilleure manière d’échapper à ce risque, c’est d’utiliser systématiquement un point-virgule pour terminer chaque instruction.

Des erreurs capitales

JavaScript est sensible à la casse, c’est-à-dire à la capitalisation des caractères. Cela signifie que les noms des variables que vous créez doivent conserver leur nom exact, majuscules et minuscules y compris, durant toute leur durée de vie. Autre implication : les noms des fonctions (y compris celles de JavaScript) doivent respecter cette règle pour qu’elles ne provoquent pas de problèmes de syntaxe.

L’une des causes d’erreur parmi les plus fréquentes, c’est le cas de la méthode `getElementById` de l’objet `Document`. Vous pourriez croire qu’écrire `getElementByID` serait plus logique, mais ce n’est pas correct !

Référencer le code avant son chargement

Le code JavaScript est normalement chargé et exécuté dans l’ordre où il apparaît dans un document (le cas des fonctions étant spécifique). Cela peut poser des problèmes si vous faites référence dans un script qui se trouve vers

l'en-tête du document à une partie du code qui est située plus loin dans celui-ci. Par exemple, le Listing 20.1 illustre un cas dans lequel l'auteur a eu pour intention de modifier le code HTML entre les balises de début et de fin d'un élément contenu dans le document, tandis que la Figure 20.2 illustre ce qui se passe quand l'erreur est détectée.

LISTING 20.1 : Attention à l'ordre dans lequel le code (ou les balises) est téléchargé.

```
<html>
<head>
  <script>
    document.getElementById("monDiv").innerHTML = "Ce div est MON div";
  </script>
</head>
<body>
  <div id = "monDiv">Ce div est VOTRE div.</div>
</body>
</html>
```

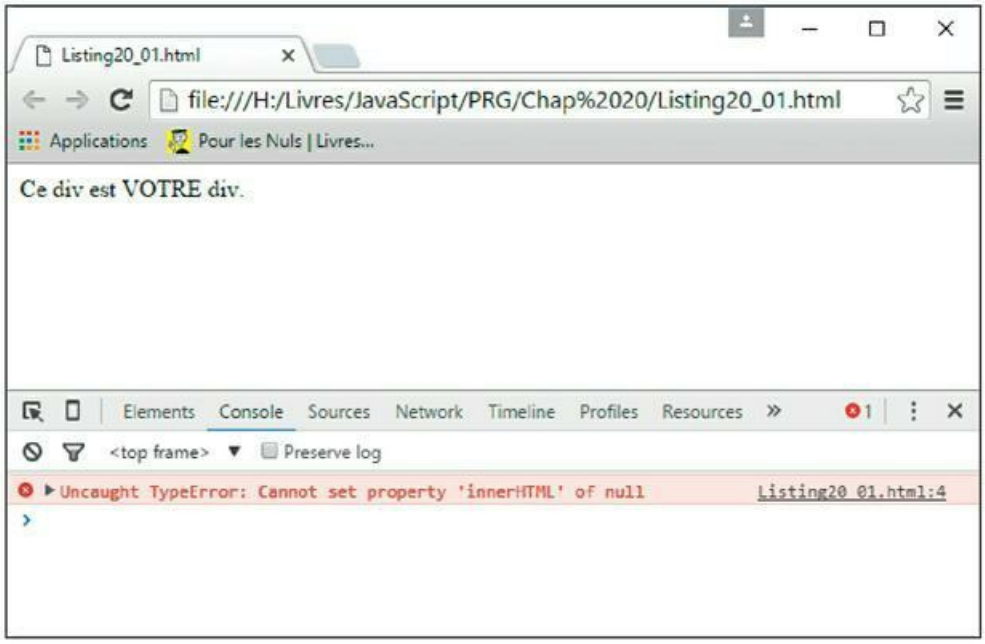


FIGURE 20.2 : Référencer du code HTML avant qu'il ne soit chargé provoque une erreur.

Ce code provoque une erreur parce qu'au moment où JavaScript est exécuté, le navigateur ne sait rien de l'élément `div` ayant pour identification `monDiv`. En effet, celui-ci n'est défini que plus loin dans la page Web.

Pour éviter ce type de problème, vous avez deux solutions :

- Placer votre code JavaScript en bas du code HTML, juste avant `</body>`.
- Placer votre code JavaScript dans une fonction. Vous appelez ensuite cette fonction en appelant l'attribut d'événement `onload` dans la balise initiale `body`.

Sur le Listing 20.2, le problème du Listing 20.1 est résolu en utilisant la seconde méthode. La Figure 20.3 montre le résultat tel qu'il est affiché dans un navigateur Web.

LISTING 20.2 : Attendez qu'une page soit totalement chargée avant d'exécuter le script.

```
<html>
<head>
    <script>
    function nameMyDiv() {
      document.getElementById("monDiv").innerHTML = "Ce div est MON div";
    }
    </script>
</head>
<body onload = "nameMyDiv();">
  <div id = "monDiv">Ce div est VOTRE div</div>
</body>
</html>
```

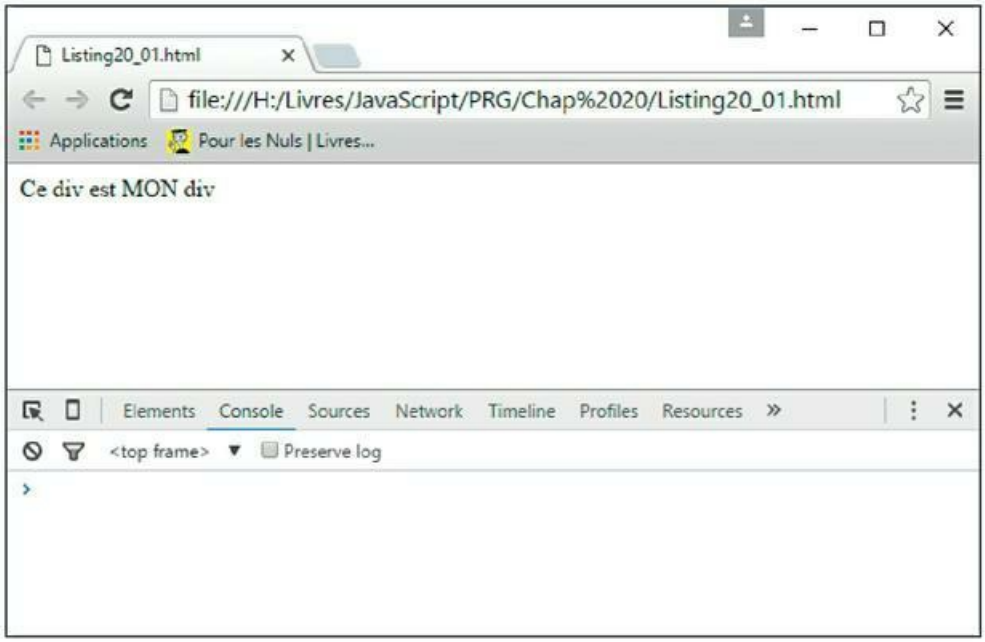


FIGURE 20.3 : Attendez que le code HTML soit entièrement chargé avant d'exécuter le script JavaScript.

De mauvais noms de variables

Les règles présidant au nommage en JavaScript sont traitées au Chapitre 3. Une particularité difficile à pister est l'interdiction d'utiliser des mots réservés comme noms de variables.

En fait, JavaScript contient plus de soixante noms réservés, et de nombreux autres que vous ne devriez pas employer pour vos variables. Plutôt que d'essayer de mémoriser tous ces mots réservés, il vaut mieux utiliser un schéma personnel descriptif qui aura peu de chances d'interférer avec les termes du langage (d'évidence, l'usage de termes de langue française pour vos noms de variables ou de fonctions devrait vous éviter ce genre de souci).

Par exemple, le mot `name` est l'un des mots réservés de JavaScript. Si vous travaillez avec des collègues dont l'anglais est la langue de base, vous aurez peut-être besoin d'échapper à ces conventions. Dans ce cas, utilisez des noms comme `firstName`, `LastName` ou encore `dogName` ou bien `nameOfTheWind` pour éviter tout conflit avec des mots réservés.

Erreurs de portée

La notion de portée en JavaScript est soit locale, soit globale. Si vous déclarez une variable sans le mot-clé `var`, sa portée est globale, et elle sera donc utilisable partout dans le programme. Comme nous l'avons montré dans le Chapitre 3, les résultats peuvent être destructifs pour votre application.

Pour éviter ce genre de problème, utilisez toujours le mot-clé `var` lorsque vous déclarez vos variables.

Paramètres manquants dans les appels de fonctions

Chaque fois que vous déclarez une fonction, vous devez aussi déclarer le nombre de paramètres que vous devriez lui passer lorsqu'elle est appelée. Se tromper dans ce nombre ne provoque pas forcément une erreur. En revanche, cela peut se traduire par des résultats inattendus si la fonction attend des paramètres qui ne sont pas présents.

Assurez-vous que vous donnez à vos paramètres des noms descriptifs lorsque vous définissez des fonctions, et vérifiez plutôt deux fois qu'une fonction est appelée dans l'ordre voulu pour les paramètres qu'elle attend afin de vous assurer que ceux -ci sont au complet.

Oublier que JavaScript compte à partir de zéro

Si vous dénombrez un tableau JavaScript en comptant jusqu'à dix, vous avez en réalité onze éléments (voir la Figure 20.4). N'oubliez donc jamais que l'indice de départ est toujours égal à zéro.

```
var myArray = new Array();
myArray[10] = "Liste des 10 erreurs courantes";
myArray.length; // produit 11 !
```

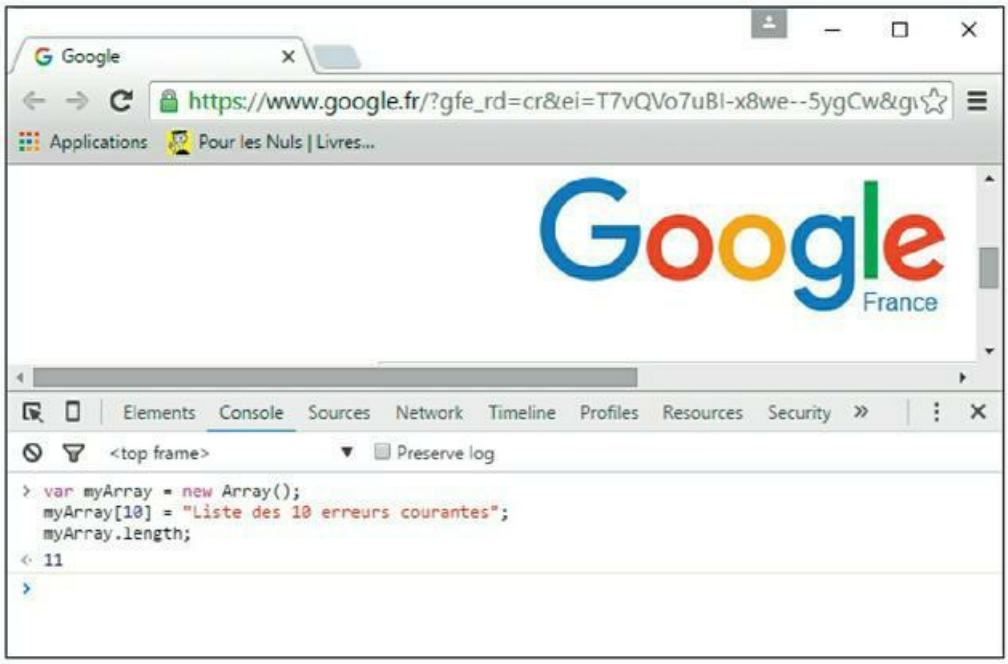


FIGURE 20.4 : Oublier que JavaScript compte à partir de zéro peut produire des résultats inattendus.

Chapitre 21

Dix outils en ligne pour vous aider à écrire du meilleur code JavaScript

DANS CE CHAPITRE :

- **Vérifier votre code avec JSLint**
- **Jouer avec JSFiddle**
- **Du code plus beau avec JSbeautifier**
- **Diminuer la taille de vos fichiers JavaScript**

« Ne sous-estimez jamais la puissance d'un simple outil. »

Craig Bruce

JavaScript possède plus de bibliothèques, de ressources et d'outils commodes que n'importe quel autre langage de programmation. Ce chapitre vous présente dix des meilleures ressources pour vous aider à écrire du meilleur code JavaScript.

JSLint

JSLint, créé par Douglas Crockford, un super génie du JavaScript, est un outil de vérification conçu pour vous dire où votre code a des problèmes, et pas juste le genre de problèmes qui pourraient générer des erreurs.

JSLint vous informe sur des choses que des milliers de programmeurs JavaScript font tout le temps, mais qui sont problématiques pour une raison ou pour une autre (voir la Figure 21.1). Si votre code passe les tests de JSLint, il est probablement déjà bon.

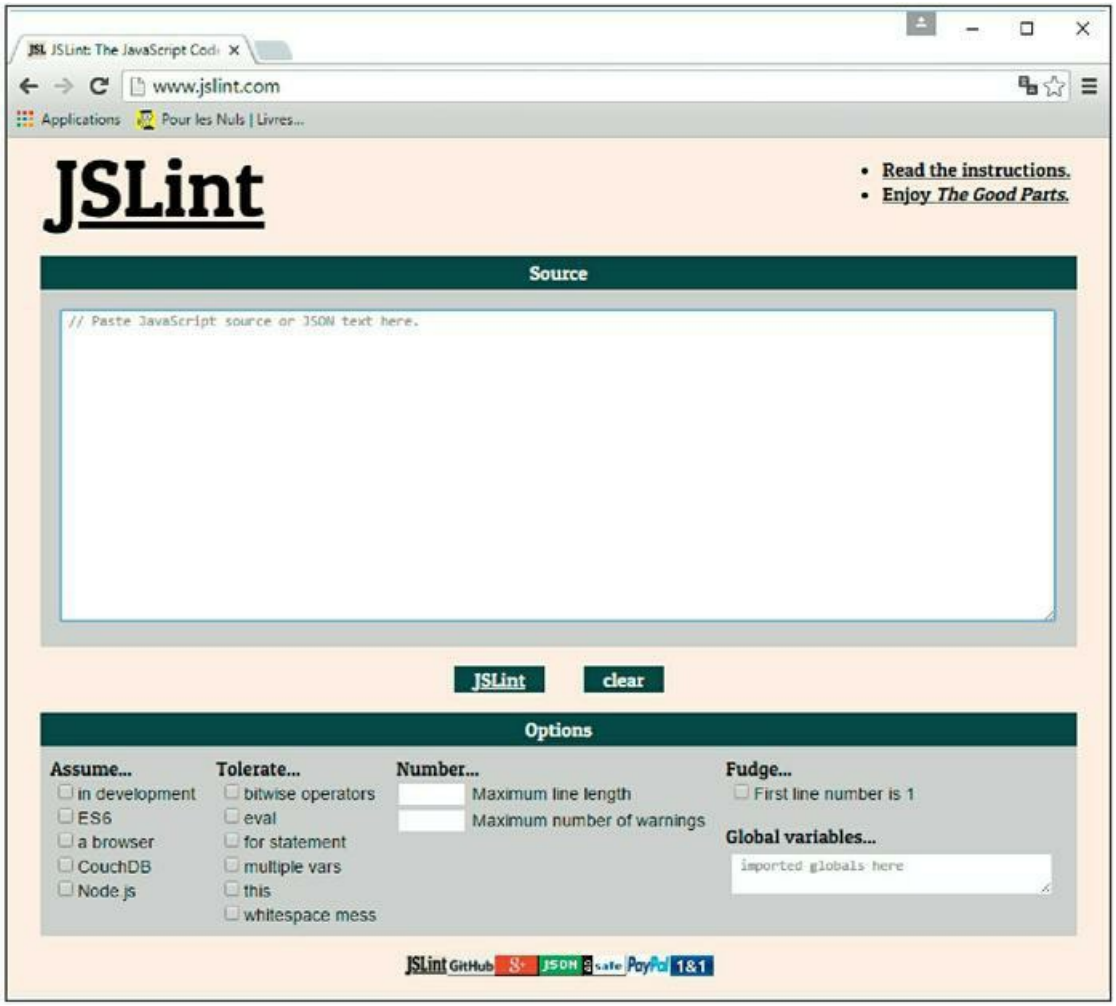


FIGURE 21.1 : JSLint vous montre les problèmes posés par votre code.

JSFiddle.net

JSFiddle est un programme en ligne servant à exécuter des applications Web dans un environnement de test (voir la Figure 21.2).

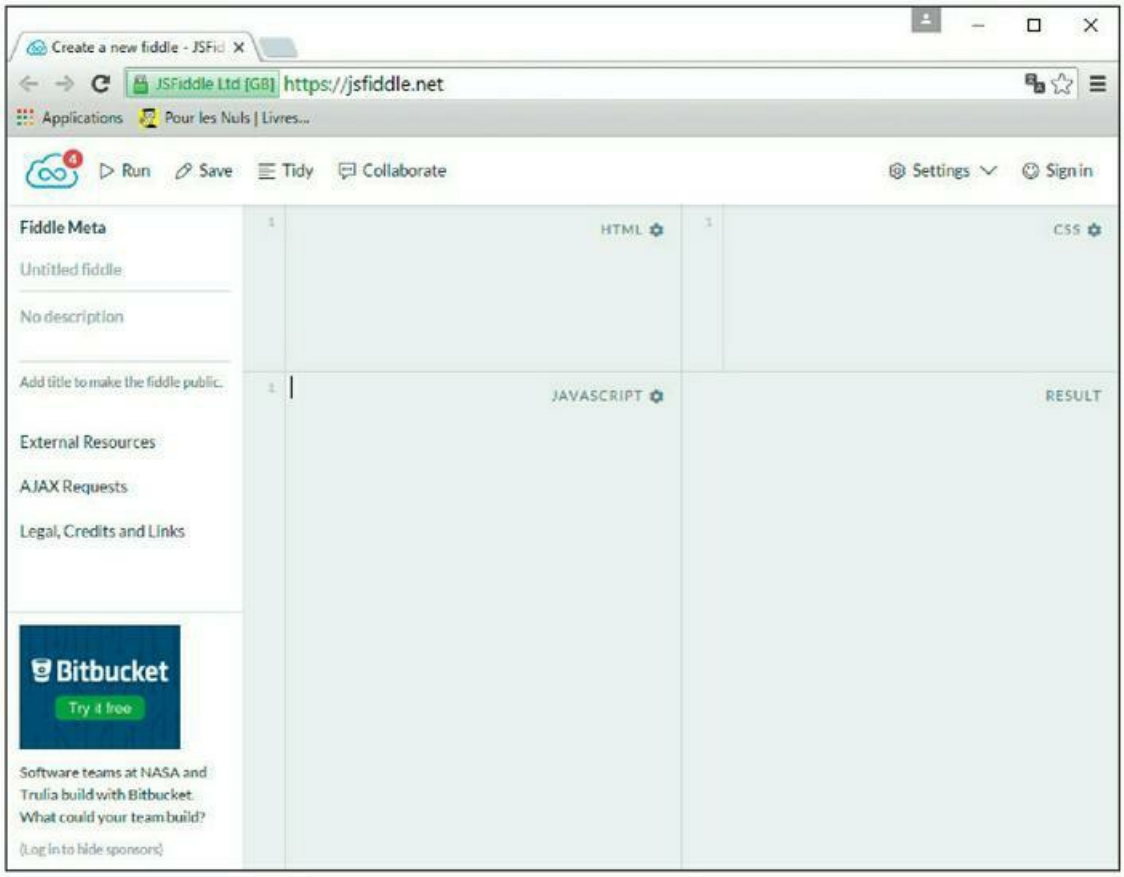


FIGURE 21.2 : JSFiddle vous permet de tester vos applications Web.

Lorsque vous accédez à JSFiddle, la première chose que vous voyez est un groupe de quatre panneaux :

- un pour le code HTML ;
- un pour les styles CSS ;
- un pour JavaScript ;
- un pour les résultats.

Entrez le type de code approprié dans les trois premiers panneaux, puis cliquez sur le bouton Run. Les résultats vont s'afficher dans le quatrième panneau.



Avec JSFiddle, vous pouvez même sauvegarder vos tests et envoyer par mail l'URL correspondante à d'autres personnes.

JSBin

JSBin est un site de partage de code qui vous permet de partager vos créations avec d'autres personnes. Vous pouvez avoir des tendances exhibitionnistes, vouloir former un jeune développeur, ou tout simplement collaborer avec d'autres programmeurs sur un projet. Dans tous les cas, les fonctionnalités de JSBin peuvent être très utiles pour effectuer votre travail sans erreurs, pour obtenir des retours d'informations, ou tout simplement pour partager du code (voir la Figure 21.3).

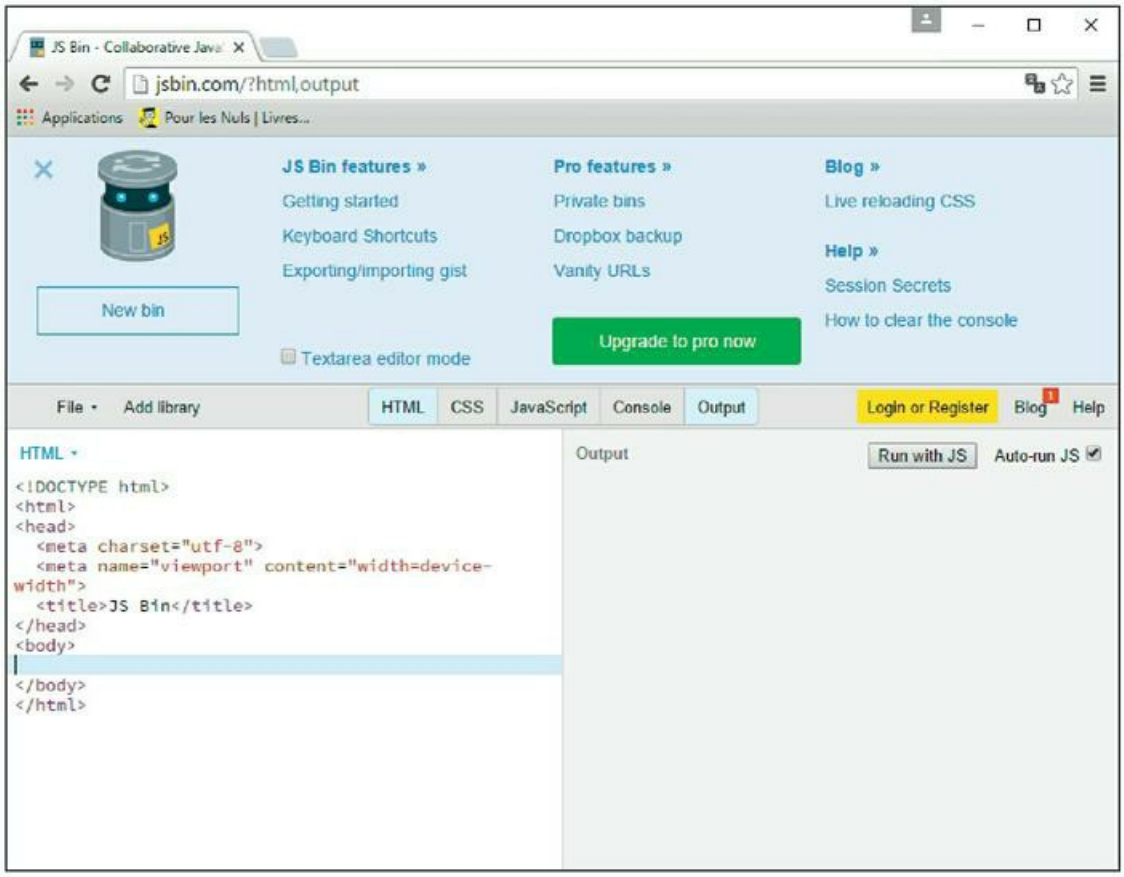


FIGURE 21.3 : Collaborer avec JSBin.

Javascriptcompressor.com

Plus vos fichiers JavaScript sont petits, et plus vite ils sont téléchargés. Le site Javascript.com propose une fenêtre dans laquelle vous pouvez déposer votre code JavaScript (voir la Figure 21.4).

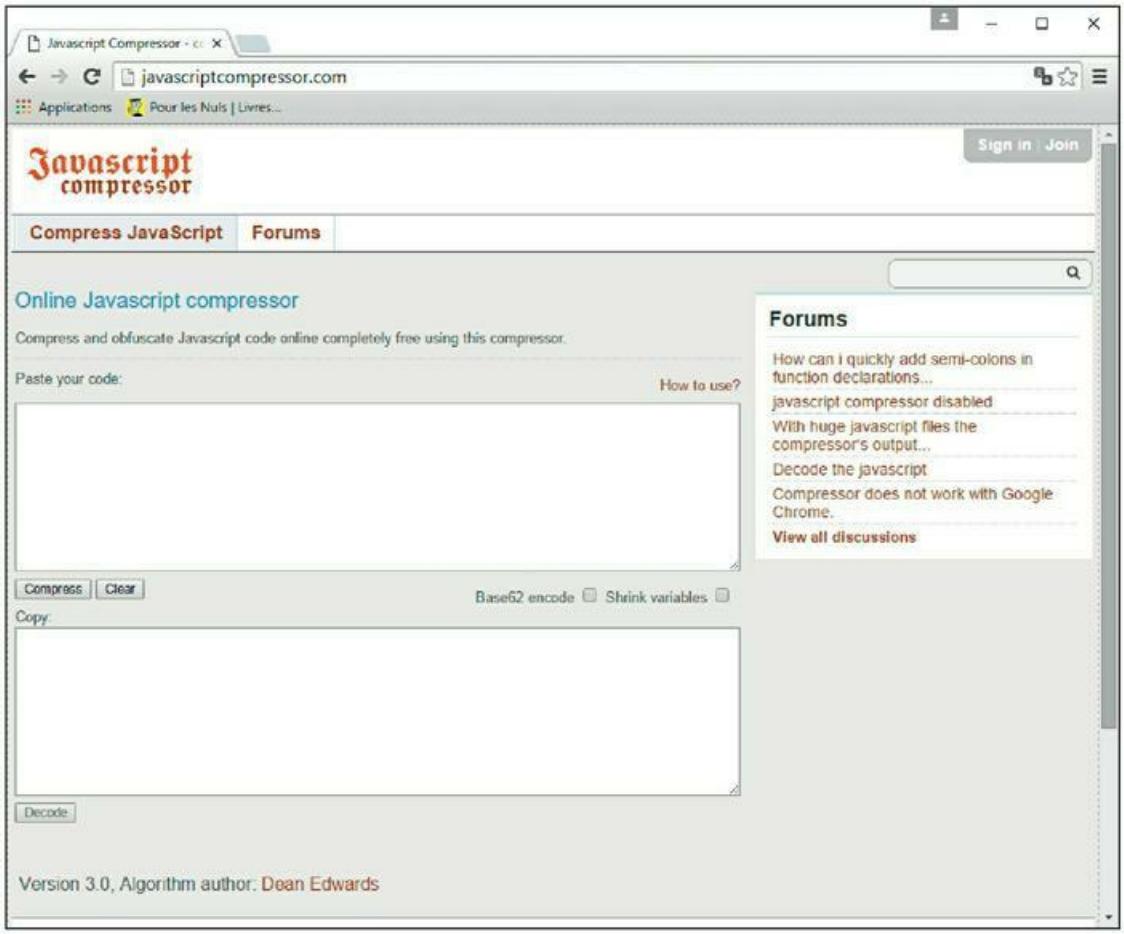


FIGURE 21.4 : Javascriptcompressor.com rend vos fichiers JavaScript plus petits.

Lorsque vous cliquez sur le bouton Compress, une nouvelle version, qui reste fonctionnellement la même que votre code original, mais sous un format plus compact, vous est proposée dans la fenêtre du bas. Ce code prend moins de place sur le disque (et donc réclame moins de bande passante), mais, en plus, il est aussi « obscurci » de manière à camoufler ses procédures secrètes devant des yeux inquisiteurs.

JSBeautifier.org

JSBeautifier est un outil en ligne qui reçoit du code JavaScript réputé être laid pour le rendre plus beau (voir la Figure 21.5).

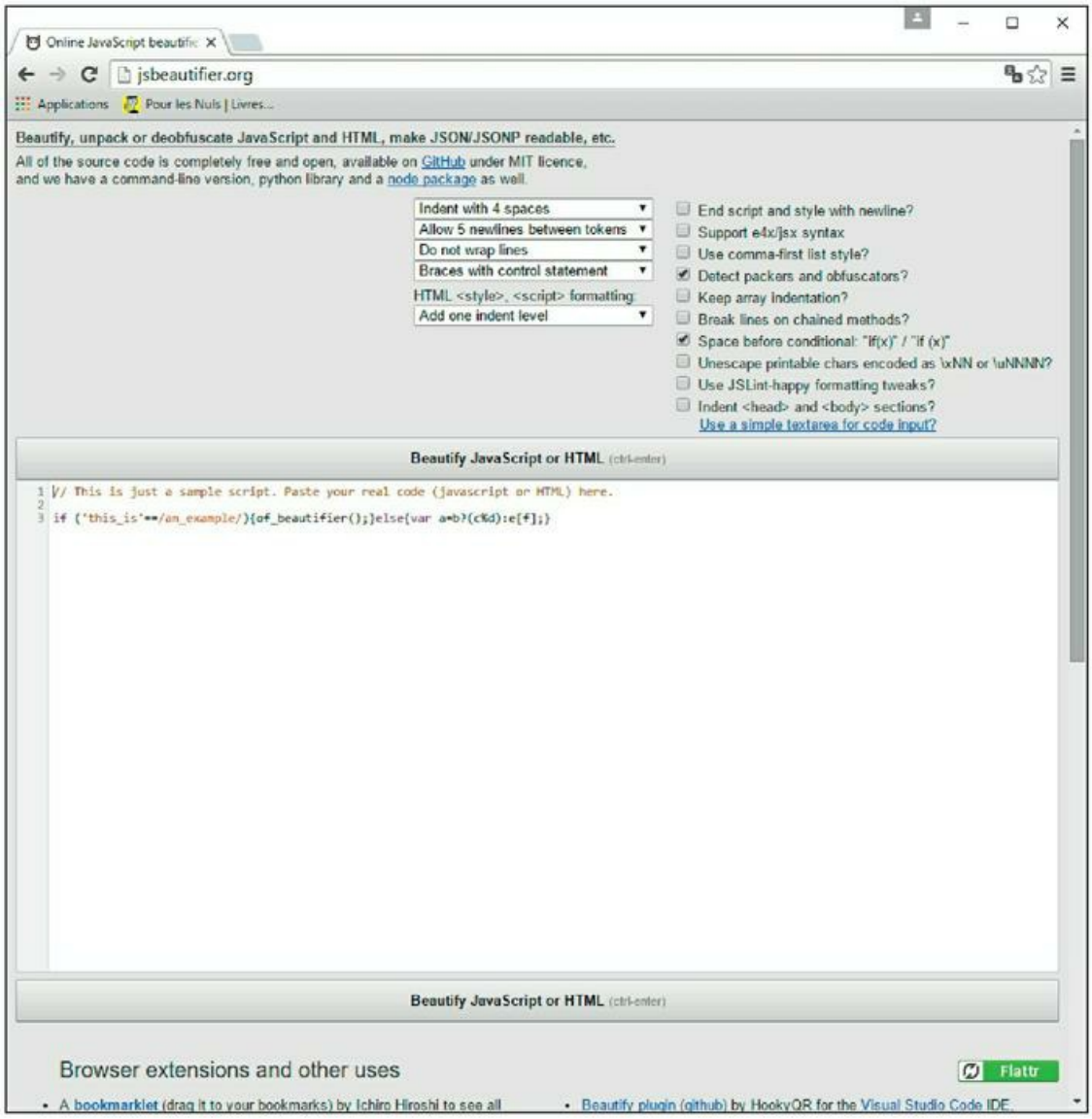

Online JavaScript beautifier
jsbeautifier.org
Applications
Pour les Nuls | Livres...
Beautify, unpack or deobfuscate JavaScript and HTML, make JSON/JSONP readable, etc.
All of the source code is completely free and open, available on GitHub under MIT licence,
and we have a command-line version, python library and a node package as well.
Indent with 4 spaces
Allow 5 newlines between tokens
Do not wrap lines
Braces with control statement
HTML <style>, <script> formatting:
Add one indent level
End script and style with newline?
Support e4x/jsx syntax
Use comma-first list style?
Detect packers and obfuscators?
Keep array indentation?
Break lines on chained methods?
Space before conditional: "if(x)" / "if (x)"
Unescape printable chars encoded as \xNN or \uNNNN?
Use JSLint-happy formatting tweaks?
Indent <head> and <body> sections?
Use a simple textarea for code input?
Beautify JavaScript or HTML (ctrl-enter)
// This is just a sample script. Paste your real code (javascript or HTML) here.
if ('this_is'==/an_example/){of_beautifier();}else{var a=b?(c%d):e[f];}
Beautify JavaScript or HTML (ctrl-enter)
Browser extensions and other uses
Flattr
A bookmarklet (drag it to your bookmarks) by Ichiro Hiroshi to see all
Beautify plugin (github) by HookyQR for the Visual Studio Code IDE

FIGURE 21.5 : Rendez votre code plus lisible avec http://jsbeautifier. org.

Parmi les techniques employées par JSBeautifier, mentionnons en particulier :

- l'insertion de nouvelles lignes ;
- la coupure de lignes présentant du code chaîné ;
- l'insertion d'espaces avant des instructions conditionnelles ;
- l'ajout d'indentations standard dans le script.

JSONFormatter

Le système de formatage et de validation JSON appelé JSONFormatter vous permet de coller du code JSON non formaté, par exemple celui que vous pourriez obtenir en le collant depuis les outils de développement de Chrome. Il améliore alors la présentation du code et vérifie qu'il est valide. JSONFormatter est accessible à l'adresse `http://jsonformatter.curiousconcept.com` (voir la Figure 21.6).

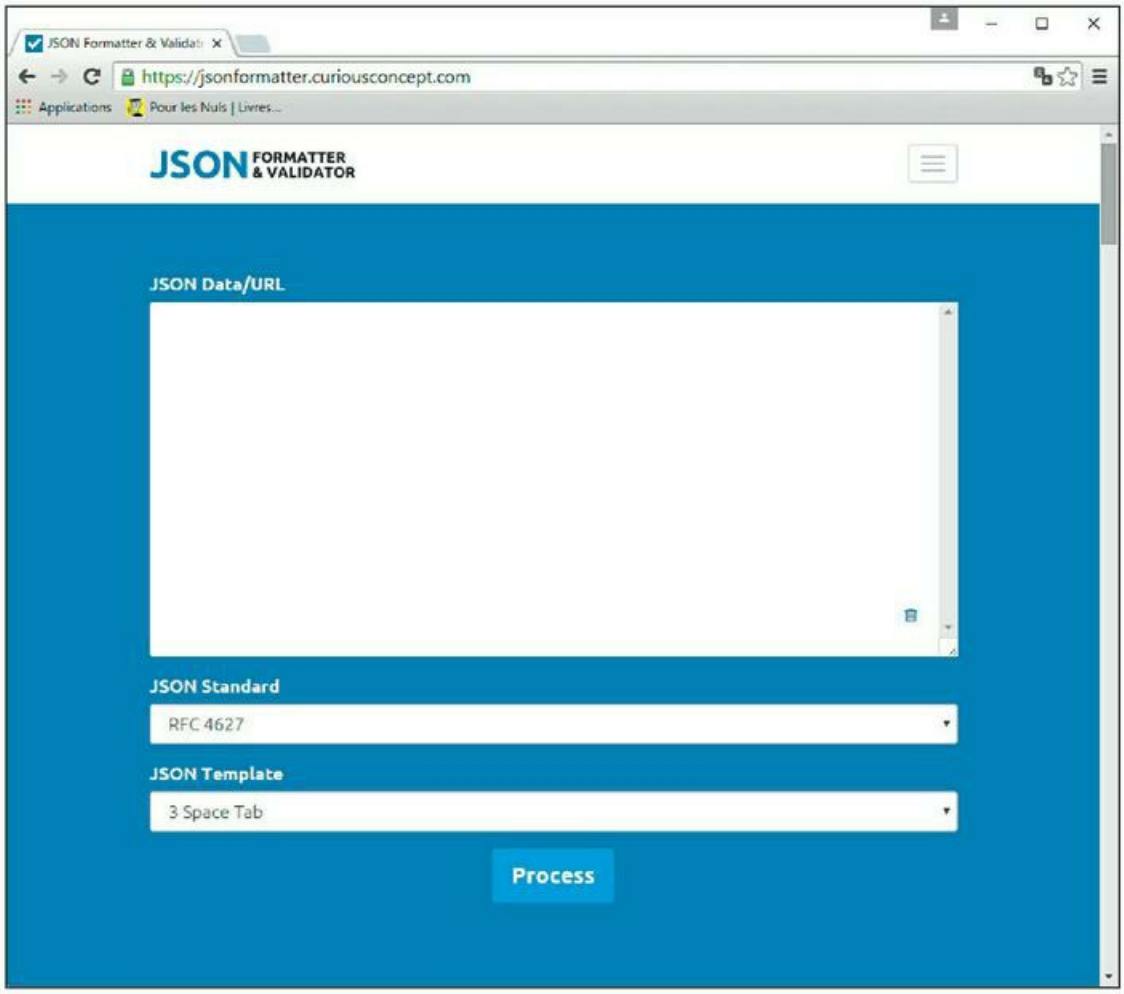


FIGURE 21.6 : JSONFormatter valide et améliore la lisibilité des données JSON.

JavaScript RegEx Generator

Le générateur JavaScript de RegEx (en théorie à l'adresse `www.jslab.dk/tools.regex.php`) est un formulaire utilisateur permettant de créer et de tester des expressions régulières. Vous cliquez simplement sur quelques boutons, vous entrez le texte à vérifier, vous définissez quelques options et vos expressions régulières apparaissent en bas de la fenêtre.



À l'heure où ces lignes sont écrites, le site `www.jslab.dk` semble ne plus être disponible. À vous de vérifier ce fait lorsque vous lirez ces lignes… Vous pouvez cependant essayer l'adresse `https://github.com/bti360/jslab` (ou `http://bti360.github.io/jslab/#/`).

Jshint.com

JShint est un outil qui vous aide à détecter des erreurs et des problèmes potentiels dans votre code JavaScript (voir la Figure 21.7). De plus, il vous fournit des informations utiles sur votre code JavaScript en même temps vous l'écrivez.

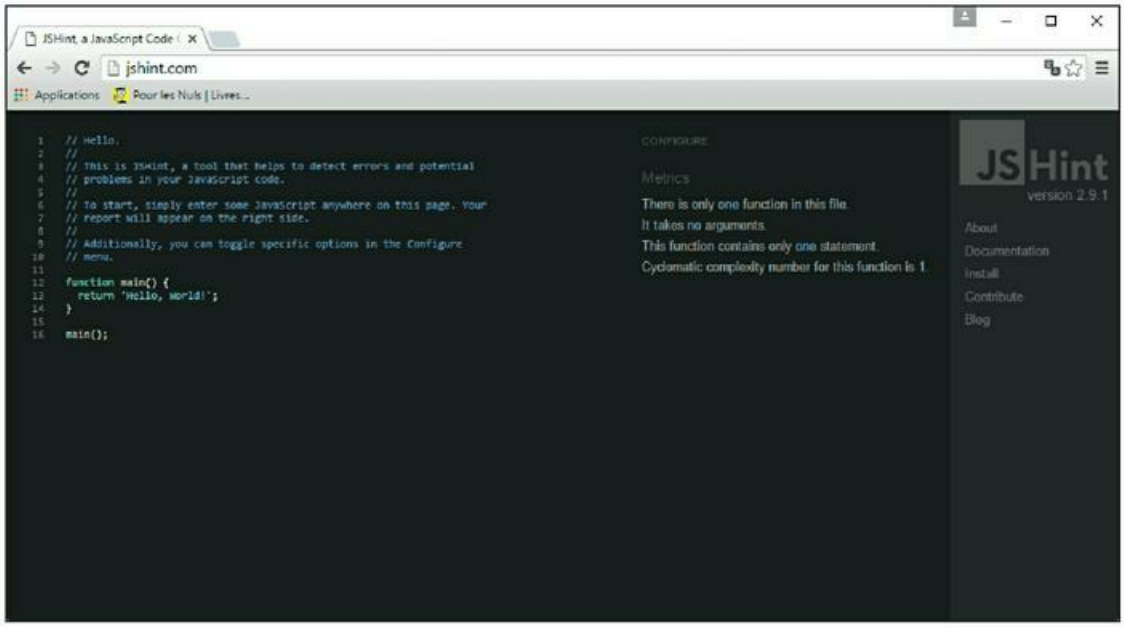


FIGURE 21.7 : JShint détecte les problèmes avec le code que vous écrivez.

Mozilla Developer Network

La section JavaScript du collectif Mozilla Developer Network (https://developer.mozilla.org/fr/docs/Web/JavaScript) est une source essentielle d'informations pour tout ce qui concerne JavaScript. Ces ressources incluent des tutoriels, des articles, des matériaux de référence, ainsi que des démonstrations, le tout étant destiné à des programmeurs de niveau différent (voir la Figure 21.8).

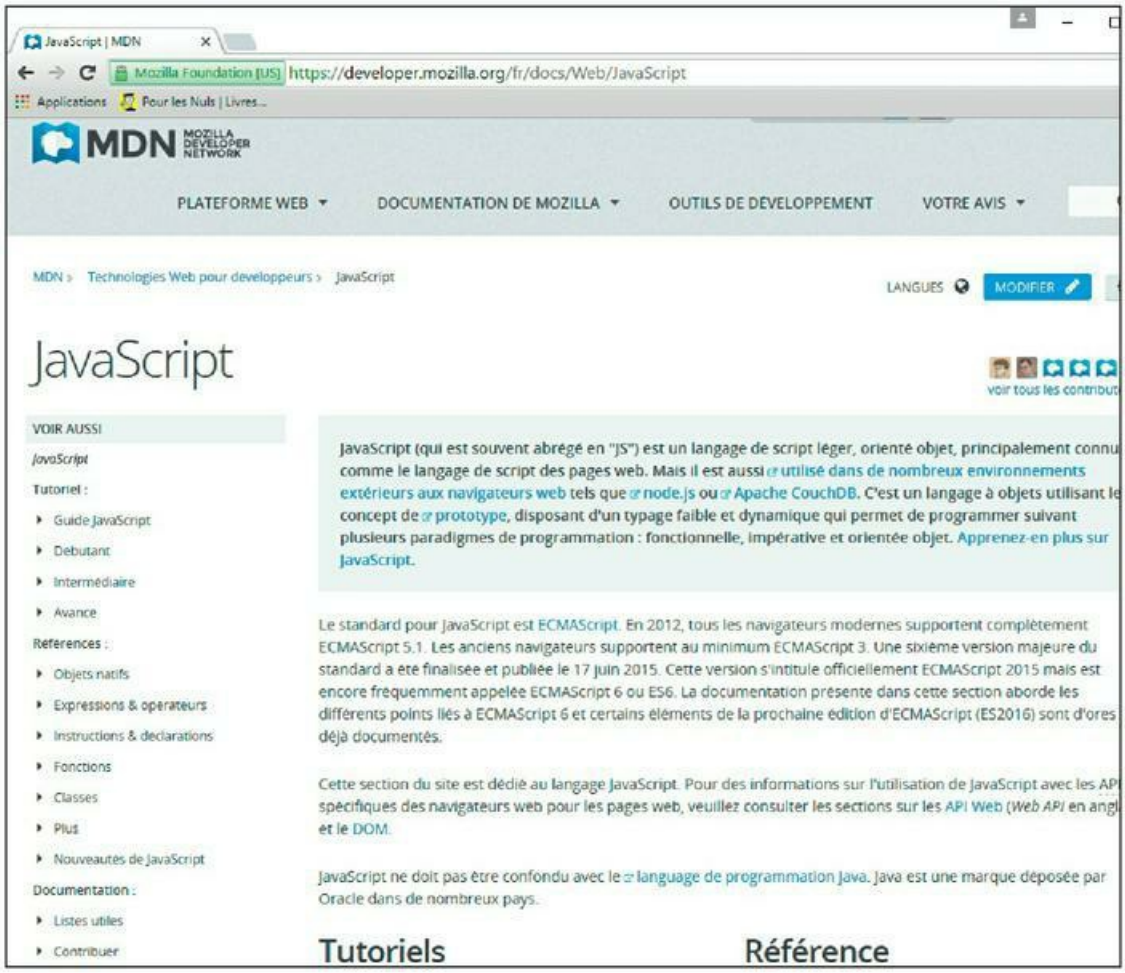

JavaScript | MDN
Mozilla Foundation (US) https://developer.mozilla.org/fr/docs/Web/JavaScript
Applications
Pour les Nuls | Livres...
MDN MOZILLA DEVELOPER NETWORK
PLATEFORME WEB
DOCUMENTATION DE MOZILLA
OUTILS DE DEVELOPPEMENT
VOTRE AVIS
MDN > Technologies Web pour developpeurs > JavaScript
LANGUES
MODIFIER
JavaScript
VOIR AUSSI
JavaScript
Tutoriel :
Guide JavaScript
Debutant
Intermediaire
Avance
References :
Objets natifs
Expressions & operateurs
Instructions & declarations
Fonctions
Classes
Plus
Nouveautés de JavaScript
Documentation :
Listes utiles
Contribuer
JavaScript (qui est souvent abrégé en "JS") est un langage de script léger, orienté objet, principalement connu comme le langage de script des pages web. Mais il est aussi utilisé dans de nombreux environnements extérieurs aux navigateurs web tels que node.js ou Apache CouchDB. C'est un langage à objets utilisant le concept de prototype, disposant d'un typage faible et dynamique qui permet de programmer suivant plusieurs paradigmes de programmation : fonctionnelle, impérative et orientée objet. Apprenez-en plus sur JavaScript.
Le standard pour JavaScript est ECMAScript. En 2012, tous les navigateurs modernes supportent complètement ECMAScript 5.1. Les anciens navigateurs supportent au minimum ECMAScript 3. Une sixième version majeure du standard a été finalisée et publiée le 17 juin 2015. Cette version s'intitule officiellement ECMAScript 2015 mais est encore fréquemment appelée ECMAScript 6 ou ES6. La documentation présente dans cette section aborde les différents points liés à ECMAScript 6 et certains éléments de la prochaine édition d'ECMAScript (ES2016) sont d'ores déjà documentés.
Cette section du site est dédié au langage JavaScript. Pour des informations sur l'utilisation de JavaScript avec les API spécifiques des navigateurs web pour les pages web, veuillez consulter les sections sur les API Web (Web API en angl et le DOM.
JavaScript ne doit pas être confondu avec le language de programmation Java. Java est une marque déposée par Oracle dans de nombreux pays.
Tutoriels
Référence

FIGURE 21.8 : Le site Mozilla Developer Network est l'une des meilleures références en ce qui concerne JavaScript.

Douglas Crockford

Pour de nombreux programmeurs en JavaScript, Douglas Crockford est un vrai héros. Son site Web (`http://javascript.crockford.com`) est une véritable encyclopédie accompagnée d'une grande quantité de vidéos couvrant chaque aspect de JavaScript (voir la Figure 21.9).

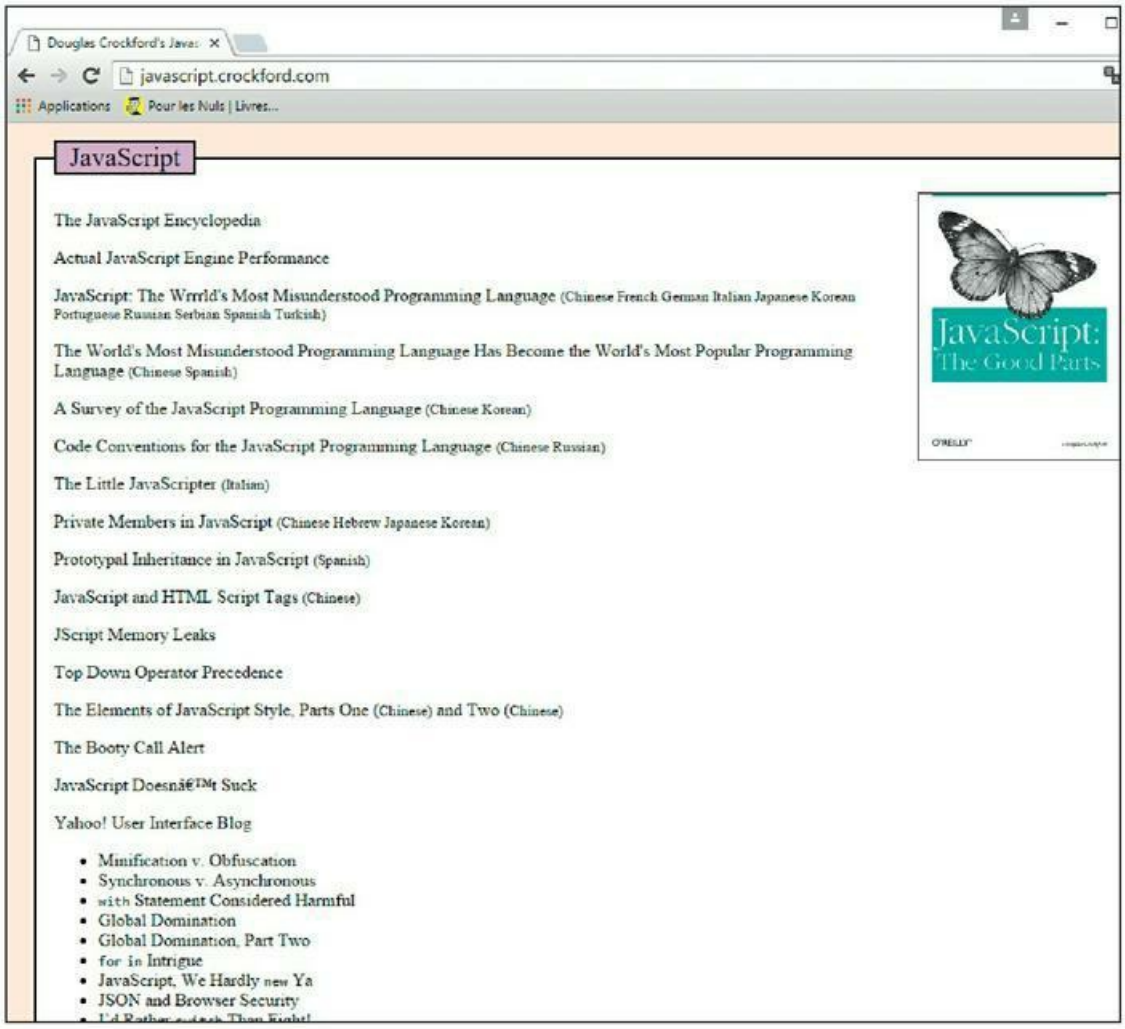

Douglas Crockford's Java:
javascript.crockford.com
Applications
Pour les Nuls | Livres...
JavaScript
The JavaScript Encyclopedia
Actual JavaScript Engine Performance
JavaScript: The Wrrrld's Most Misunderstood Programming Language (Chinese French German Italian Japanese Korean Portuguese Russian Serbian Spanish Turkish)
The World's Most Misunderstood Programming Language Has Become the World's Most Popular Programming Language (Chinese Spanish)
A Survey of the JavaScript Programming Language (Chinese Korean)
Code Conventions for the JavaScript Programming Language (Chinese Russian)
The Little JavaScripter (Italian)
Private Members in JavaScript (Chinese Hebrew Japanese Korean)
Prototypal Inheritance in JavaScript (Spanish)
JavaScript and HTML Script Tags (Chinese)
JScript Memory Leaks
Top Down Operator Precedence
The Elements of JavaScript Style, Parts One (Chinese) and Two (Chinese)
The Booty Call Alert
JavaScript Doesnâ€™t Suck
Yahoo! User Interface Blog
Minification v. Obfuscation
Synchronous v. Asynchronous
with Statement Considered Harmful
Global Domination
Global Domination, Part Two
for in Intrigue
JavaScript, We Hardly new Ya
JSON and Browser Security
JavaScript:
The Good Parts
O'REILLY

FIGURE 21.9 : Le site de Douglas Crockford propose des quantités de vidéos pour les programmeurs JavaScript.

Ces vidéos sont essentielles pour un programmeur qui veut dépasser le stade du débutant pour progresser vers des niveaux plus avancés d'expertise en JavaScript.

Sommaire

Couverture

Javascript 2e édition Pour les Nuls

Copyright

Introduction

- À propos de ce livre
- Quelques suppositions stupides
- Icônes utilisées dans ce livre
- Où trouver les exemples du livre ?
- Et maintenant ?

I. Débuter avec JavaScript

- Chapitre 1. JavaScript, le langage le plus incompris du monde
  - JavaScript, c'est quoi ?
- Chapitre 2. Écrire votre premier programme JavaScript
  - Configurer votre environnement de développement
  - Lire le code JavaScript
  - Exécuter JavaScript dans la fenêtre du navigateur
  - Utiliser la console de développement JavaScript
  - Commenter votre code
- Chapitre 3. Travailler avec les variables
  - Comprendre les variables
  - Déclarer des variables
  - Comprendre la portée des variables
  - Nommer les variables
  - Créer des constantes avec le mot-clé const
  - Travailler avec les types de données
- Chapitre 4. Comprendre les tableaux
  - Faire une liste
  - Notions de base sur les tableaux
  - Créer des tableaux
  - Remplir les tableaux
  - Comprendre les tableaux multidimensionnels
  - Accéder aux éléments d'un tableau

Chapitre 5. Travailler avec les opérateurs, les expressions et les instructions

Exprimez-vous

Bonjour, opérateur

Types d'opérateurs

Chapitre 6. Rester dans le flux avec les boucles et les branchements

Soyez branché !

Boucler les boucles

II. Organiser votre code JavaScript

Chapitre 7. Devenir fonctionnel

Comprendre la fonction des fonctions

Utiliser la terminologue des fonctions

Les fonctions, c'est tout bénéfice !

Écrire des fonctions

Retourner des valeurs

Passer et utiliser des arguments

Portée des fonctions

Le cas de la fonction anonyme

On recommence depuis le début (la récursivité)

Des fonctions dans des fonctions

Chapitre 8. Créer et utiliser des objets

L'obscur objet de mon désir…

Créer des objets

Retrouver et définir les propriétés des objets

Supprimer des propriétés

Travailler avec les méthodes

Une méthode orientée objet pour devenir riche : l'héritage

III. JavaScript sur le Web

Chapitre 9. Contrôler le navigateur avec l'objet Window

Comprendre l'environnement du navigateur

Chapitre 10. Manipuler des documents avec le DOM

Comprendre le DOM

Relations entre les nœuds

Utiliser les propriétés et les méthodes de l'objet Document

Utiliser les propriétés et les méthodes de l'objet Element

Travailler avec le contenu des éléments

Trouver des éléments par ID, nom de balise ou classe

Utiliser les propriétés de l'objet Attribut

Chapitre 11. Utiliser des événements dans JavaScript

Connaître vos événements

Gérer les événements

Chapitre 12. Entrer et sortir

Comprendre les formulaires HTML

Travailler avec l'objet Form

Chapitre 13. Travailler avec CSS et les graphismes

Utiliser l'objet Style

Animer des éléments avec l'objet Style

Travailler avec les images

Utiliser les propriétés d'animation de l'objet Style

IV. Au-delà des bases

Chapitre 14. Effectuer des recherches avec des expressions régulières

Faire des recherches avec les expressions régulières

Écrire des expressions régulières

Utiliser des modificateurs

Coder avec des expressions régulières

Chapitre 15. Comprendre les fonctions de rappel et les fermetures

C'est quoi, les fonctions de rappel ?

Comprendre les fermetures

Utiliser les fermetures

Chapitre 16. Toujours plus loin avec Ajax et JSON

Travailler en coulisses avec Ajax

Mettre des objets en mouvement avec JSON

V. JavaScript et HTML5

Chapitre 17. HTML5 et ses API

Comprendre comment les API fonctionnent

Utiliser la géolocalisation

Accéder à l'audio et à la vidéo

Chapitre 18. jQuery

Écrire plus, tout en en faisant moins

Débuter avec jQuery

L'objet jQuery

Est-ce que votre document est prêt ?

Utiliser les sélecteurs jQuery

Changer des choses avec jQuery

Événements

Effets

Ajax

VI. Les Dix Commandements

Chapitre 19. Dix bibliothèques et frameworks pour aller plus loin avec JavaScript

Angular JS

Backbone.js

Ember.js

Famo.us

Knockout

QUnit

Underscore.js

Modernizr

Handlebars.js

jQuery

Chapitre 20. Dix erreurs JavaScript courantes et comment les éviter

Confusion dans l'égalité

Les crochets mal accrochés

Confusion entre guillemets et apostrophes

Les parenthèses manquantes

Le point-virgule manquant

Des erreurs capitales

Référencer le code avant son chargement

De mauvais noms de variables

Erreurs de portée

Paramètres manquants dans les appels de fonctions

Oublier que JavaScript compte à partir de zéro

Chapitre 21. Dix outils en ligne pour vous aider à écrire du meilleur code JavaScript

JSLint

JSFiddle.net

JSBin

Javascriptcompressor.com

JSBeautifier.org

JSONFormatter

JavaScript RegEx Generator

Jshint.com

Mozilla Developer Network

Douglas Crockford